

Software Testing
and Continuous Quality Improvement

Third Edition

软件测试 与持续质量改进

(第3版)

William E. Lewis

[美] David Dobbs

Gunasekaran Veerapillai

陈绍英 张河涛 刘建华 金成姬 等译

冯国云 尹 众 审校



人民邮电出版社

POSTS & TELECOM PRESS

图书在版编目 (CIP) 数据

软件测试与持续质量改进 : 第3版 / (美) 刘易斯 (Lewis, W. E.), (美) 多布斯 (Dobbs, D.), (美) 维拉皮莱 (Veerapillai, G.) 著 ; 陈绍英等译. -- 北京 : 人民邮电出版社, 2011.7

(图灵程序设计丛书)

书名原文: Software Testing and Continuous Quality Improvement

ISBN 978-7-115-25540-2

I. ①软… II. ①刘… ②多… ③维… ④陈… III. ①软件—测试 IV. ①TP311.5

中国版本图书馆CIP数据核字(2011)第101543号

内 容 提 要

本书为软件测试过程提供了一个质量框架, 目的是提出一个持续改进软件质量的途径, 以提高测试效率。书中详细列举基本的软件测试技巧, 并基于一种持续改进过程介绍 Deming 的质量概念, 将“计划、执行、检查、改进”(Plan, Do, Check, Act, PDCA) 这样一个质量循环引入软件测试过程, 阐述现代质量保证理论及最佳实践方法。此外, 附录中提供软件测试过程中可能涉及的各种文档的格式样本, 非常便于查阅和参考。

本书既适合软件测试领域的专业技术人员作为参考手册, 又适合作为计算机及相关专业软件测试课程的教材。

图灵程序设计丛书

软件测试与持续质量改进 (第3版)

◆ 著 [美] William E. Lewis David Dobbs Gunasekaran Veerapillai
译 陈绍英 张河涛 刘建华 金成姬 等
审 校 冯国云 尹 众
责任编辑 杨海玲

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷

◆ 开本: 800×1000 1/16
印张: 29.25
字数: 741千字 2011年7月第1版
印数: 1-3 000册 2011年7月北京第1次印刷

著作权合同登记号 图字: 01-2009-5709号

ISBN 978-7-115-25540-2

定价: 75.00元

读者服务热线: (010)51095186转604 印装质量热线: (010)67129223

反盗版热线: (010)67171154

目 录

第一部分 软件质量透视

第 1 章 软件测试简史	2
1.1 历史上软件测试和开发并行	4
1.2 极限编程	5
1.3 自动化测试工具的发展	6
1.3.1 静态捕获/回放工具（不附带脚本语言）	7
1.3.2 静态捕获/回放工具（具有脚本语言）	7
1.3.3 可变的捕获/回放工具	7
第 2 章 质量保证框架	9
2.1 什么是质量	9
2.2 预防与检测	10
2.3 验证与确认	10
2.4 软件质量保证	11
2.5 质量保证的组成	12
2.5.1 软件测试	12
2.5.2 质量控制	13
2.5.3 软件配置管理	14
2.6 软件质量保证计划	16
2.7 质量标准	18
2.7.1 萨班斯-奥克斯利法案	18
2.7.2 ISO9000	19
2.7.3 能力成熟度模型	20
2.7.4 PCMM	23
2.7.5 CMMI	23
2.7.6 美国波多里奇国家质量奖	23
2.8 注释	25

第 3 章 测试技术概述	26
3.1 黑盒测试（功能测试）	26
3.2 白盒测试（结构测试）	27
3.3 灰盒测试（功能与结构测试）	27
3.4 手工测试与自动化测试	27
3.5 静态测试与动态测试	27
3.6 软件测试技术的分类	28
第 4 章 将需求转换成可测试的测试用例	33
4.1 概述	33
4.2 将软件需求作为测试的基础	33
4.3 需求质量因素	34
4.3.1 可理解	34
4.3.2 必需	34
4.3.3 可修改	34
4.3.4 非冗余	34
4.3.5 简洁	35
4.3.6 可测试	35
4.3.7 可跟踪	35
4.3.8 在范围内	35
4.4 评估需求质量的数值方法	35
4.5 根据好的需求创建测试用例的过程	36
4.5.1 步骤1：评审需求	36
4.5.2 步骤2：编写测试计划	38
4.5.3 步骤3：确定测试套件	38
4.5.4 步骤4：命名测试用例	40
4.5.5 步骤5：编写测试用例描述及目标	41

第二部分 瀑布测试概述

第 7 章 需求的静态测试	72
7.1 通过歧义性评审测试需求	73
7.2 通过技术评审测试需求	73
7.3 审查和走查	73
7.4 检查表	74
7.5 需求可追溯性矩阵	74
7.6 制定系统/验收测试计划	74
第 8 章 逻辑设计的静态测试	77
8.1 数据模型、过程模型及其联系	77
8.2 通过技术评审测试逻辑设计	78
8.3 细化系统/验收测试计划	79
第 9 章 物理设计的静态测试	80
9.1 通过技术评审测试物理设计	80
9.2 创建集成测试用例	81
9.3 集成测试方法	81
9.3.1 步骤1: 标识出单元接口	81
9.3.2 步骤2: 全面协调接口	81
9.3.3 步骤3: 创建集成测试条件	82
9.3.4 步骤4: 评估集成测试条件的完整性	82
第 10 章 程序单元设计的静态测试	83
10.1 通过技术评审测试程序单元设计	83
10.1.1 顺序结构	83
10.1.2 选择结构	83
10.1.3 循环结构	83
10.2 编写单元测试用例	84
第 11 章 代码的静态测试与动态测试	85
11.1 用技术评审测试编码	85
11.2 执行测试计划	86
11.3 单元测试	86
11.4 集成测试	87
11.5 系统测试	87
11.6 验收测试	87
11.7 缺陷记录	88

第三部分 螺旋(敏捷)软件测试方法: 计划、执行、检查、改进

第 12 章 开发方法概述	91
12.1 生命周期开发的局限性	91
12.2 客户/服务器架构的挑战	92
12.3 客户/服务器架构中螺旋测试的心理学	93
12.3.1 新思想	93
12.3.2 对测试人员/开发人员的理解	93
12.3.3 项目的目标: 把质量保证和开发结合起来	94
12.3.4 迭代/螺旋式开发方法	94
12.4 JAD的角色	96
12.5 原型法的作用	96
12.6 开发原型的方法	97
12.6.1 步骤1: 开发原型	97
12.6.2 步骤2: 向管理层演示原型	98
12.6.3 步骤3: 向用户演示原型	98
12.6.4 步骤4: 修订并定稿规约	99
12.6.5 步骤5: 开发产品系统	99
12.7 持续改进螺旋测试方法	100
第 13 章 信息收集(计划)	103
13.1 步骤1: 准备访谈	104
13.1.1 任务1: 确定参加访谈的人	104
13.1.2 任务2: 确定议程	104
13.2 步骤2: 执行访谈	104
13.2.1 任务1: 理解项目	105
13.2.2 任务2: 理解项目目标	105
13.2.3 任务3: 理解项目状态	106
13.2.4 任务4: 理解项目计划	107
13.2.5 任务5: 理解项目开发方法	107
13.2.6 任务6: 确定总体业务需求	108
13.2.7 任务7: 进行风险分析	108
13.3 步骤3: 总结访谈成果	110
13.3.1 任务1: 总结访谈	110
13.3.2 任务2: 确认访谈成果	110

软件质量透视

软件测试一般被看做是“找错”。作者相信软件测试的目标是，通过根据期望和实用的标准来测量软件的属性和能力而对软件程序的质量进行量化。软件测试还为软件开发工作提供非常有价值的信息。

软件质量是所有人都想要的。经理们知道他们想要高质量，软件开发人员知道他们想要生产出一个高质量的产品，用户也坚持软件应当能始终如一地工作并且十分可靠。

许多软件质量小组做出与测试计划相似的软件质量保证计划。一个软件质量保证计划可能包括更多的活动，而不会局限于测试计划中所包括的那些活动。尽管质量保证计划包括了整个质量策略，但测试计划仍是质量保证计划的质量控制工具之一。

本部分的目标如下：

- 定义质量及其成本；
- 区分质量预防和质量检测；
- 区分验证和确认；
- 简要描述质量保证的组成；
- 简要描述通用的测试技术；
- 描述持续改进过程在获取质量的过程中是如何实现的；
- 描述软件测试的历史。

现代测试工具正在变得越来越先进，易用性也越来越出色。下面的讨论会描述随着时间推移，软件测试领域是如何发展的。从中可以看出自动化测试工具的发展前景。

软件测试是在软件源代码开发完成之后执行软件程序的活动。进行软件测试是为了在产品交付到用户手中之前，发现并修正尽可能多的潜在缺陷。正如前面所指出的，软件测试现在仍然是一门“艺术”。它可以被看做是一种风险管理技术；例如，质量保证技术是修正规约、设计或代码中所产生的偏离缺陷的最后一道防线。

在软件开发的整个历史过程中，软件测试有了许多新的定义和进步。图1-1阐述了这些演进。在20世纪50年代，软件测试被定义为“程序员为了在他们的程序中找到bug所做的事情”。在60年代早期，测试的定义得以修订，人们开始考虑对软件进行彻底的测试，以遍历代码的可能路径或全部列举可能的输入变量的方式来进行。大家注意到，完全彻底地测试一个应用程序是不可能的，一是因为程序的输入域太大，二是因为有太多可能的输入路径，三是因为设计和规约问题是很难测试的。鉴于以上这些观点，彻底的测试是要打折扣的，而且是理论上不可行的。

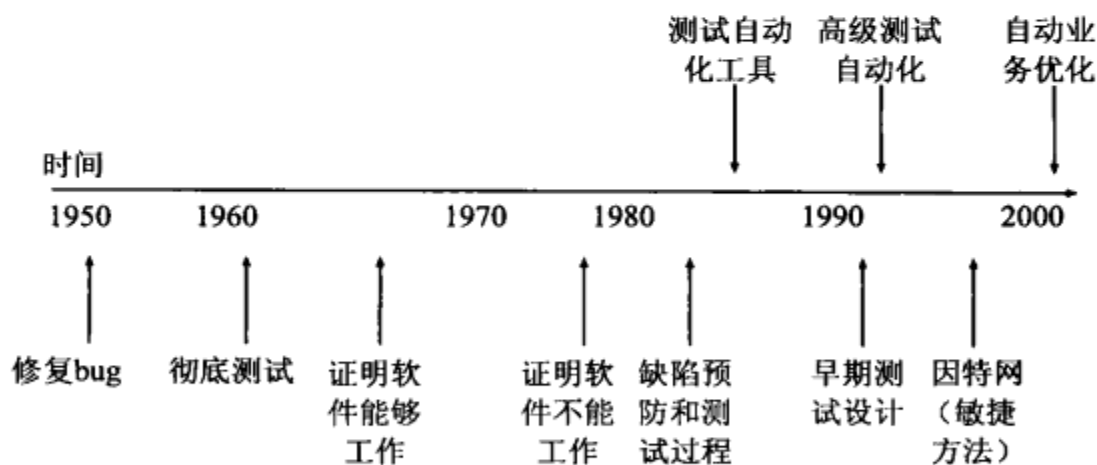


图1-1 软件测试的历史

随着20世纪60年代和70年代软件开发的成熟，软件开发活动被称为“计算机科学”。在20世纪70年代早期，软件测试被定义为“证明一个程序的正确性而要做的事情”或“对一个程序或系统做了它应该做的事情建立信心的过程”。一种短命的计算机科学技术是建议在一个软件系统的规约、设计和实现过程中通过“正确性证明”而进行软件正确性验证。尽管这个概念在理论上很有前途，但在实践中它过于耗费时间，效率极低。对简单测试而言，要证明软件能够运行并证实

它在理论上也能够运行是十分容易的。但是，由于大多数软件都不能采用这种方法进行测试，所以在实际的实现过程中还是会发现许多缺陷。大家很快便得出结论，正确性证明是一种效率低下的软件测试方法。然而，直到今天我们仍然需要一些正确性证明，例如本书中多次描述过的验收测试。

在20世纪70年代后期，测试被阐述为带着找到缺陷的意图执行程序的过程，而不是证明程序能够运行的过程。这种观点强调了好的测试用例应该有很大几率去发现尚未发现的缺陷。成功的测试是发现了尚未发现的缺陷的测试。这种方法与上面提到的观点完全相反。

上面两种测试的定义（证明能运行和证明不能运行）通过两个互相支持而又矛盾的目标给出了“测试自相矛盾”的特点。

(1) 为产品提供能够良好运行的信心。

(2) 在软件产品交付到用户手中（或下一阶段的开发）之前发现其中的缺陷。

如果第一个目标是证明一个程序能运行，则肯定会造成“我们应该下意识地向着这个目标前进；也就是说，我们应该倾向于选择造成程序失败的概率较低的测试数据”的想法。

4

如果第二个目标是在软件产品中发现缺陷，那么，尤其是在刚刚证明了这个产品实际上不能运行的情况下，如何证明这个产品能够良好运行呢？今天测试人员广泛接受的观点是：第二个目标比第一个目标要更有效率，因为如果接受了第一个目标，测试人员就会下意识地忽视一些缺陷，试图证明程序能够运行。

下面是一些好的测试原则的建议。

- 测试用例的一个必要部分是对预期输出或结果的定义。
- 应该避免让程序员测试自己编写的程序。
- 彻底全面地审查每一个测试结果。
- 除了有效和预期的输入条件外，也应该为无效和预期之外的输入条件编写测试用例。
- 检查一个程序看看它是否没有做到它应该做到的事情只是工作的一半，另外一半是要看程序是否做了它不应该做的事情。
- 要避免使用一次性的测试用例，除非程序真的是一次性的。
- 不要在默认假设不会发现缺陷的情况下计划测试工作量。
- 在一个程序的一个环节中存在更多缺陷的可能性与这个环节中已经发现的缺陷的数目是成正比的。

20世纪80年代，测试的定义扩展到包括了缺陷预防。设计测试是已知的最为有效的缺陷预防技术。最好能有一个明确的测试方法，测试要包括对整个软件开发生命周期的评审，这个评审过程应当是可管理的。测试的重要性不仅仅在于程序本身，还被扩展到需求、设计、编码、测试本身和程序上。

传统的“测试”（直到20世纪80年代早期）指的是在交付了可运行代码以后执行一个系统的过程（现在这被称为系统测试）。然而，今天的测试是“更大的测试”，测试者涉及几乎软件开发生命周期中的任一方面。一旦代码被交付测试，它将被测试和检查，如果出现了任何缺陷，都要对之前的开发阶段进行调查。这样不管缺陷是由设计不明确造成的，还是由程序员的疏忽造成的，在缺陷发生时马上找到问题的原因要比等到实际运行产品生产出来再找容易得多。研究表明，大

5

约50%的缺陷是在需求阶段（我们希望这个软件做什么？）或设计阶段造成的，这些缺陷会产生混合效应，并在代码编写过程中产生更多的缺陷。缺陷或问题在生命周期中发现得越早，修补成本就越低（数量是指数趋势）。需求或设计应该严格检查，这比测试一个软件，在其中寻找缺陷强得多。不幸的是，就是在今天，许多软件开发组织都相信软件测试是一个后备活动。

在20世纪80年代中期，自动化测试工具出现了。与手工测试相比，自动化的引入大大提高了目标应用程序的效率和质量。人们期望计算机能够比人工手动完成更多的应用程序测试，并具有更高的可靠性。这些工具起初还是相当原始的，不具有高级脚本语言工具（更多细节，请参见1.3节）。

在20世纪90年代早期，大家认识到了早期测试设计的威力。测试被重新定义为“计划、设计、创建、维护和执行测试及测试环境”。这是测试的一个质量保证的观点，认为好的测试过程是受管理的，是一个与可测试性有关的整个生命周期。

在20世纪90年代早期，更多先进的捕获/回放测试工具提供了丰富的脚本语言和报告功能。测试管理工具帮助管理从需求和测试设计到测试脚本和测试缺陷的所有内容。也有一些可用的性能工具来测试系统性能。这些工具测试已经过压力和负载测试的目标系统，以确定它们的断点。容量计划促成了这些测试。

在20世纪90年代中期，尽管人们仍坚持认为测试应该是一个贯穿整个软件开发生命周期的过程，但是随着因特网的流行，软件通常不再是在某个特定测试标准模型下开发的，这使得软件测试变得越来越困难。正如在没有明确定义每一个步骤的预期结果的条件下评审文档一样，测试通常也在没有明确预先定义所有要测试方面的情况下进行。这种测试方法被认为是“敏捷测试”。这种测试技术包括探索测试、快速测试和基于风险的测试等。

在21世纪早期，Mercury Interactive公司在他们引入业务技术优化（BTO）的概念时引入了一个更为广泛的测试定义。BTO根据业务目标调整IT策略和执行。它有助于对IT优先权、人员和过程的管理。基本方法是量度整个IT服务交付生命周期中的价值，使其最大化，以确保应用程序达到质量、性能和可用性的目标。交互数字战场实时给出了至关重要的业务可用性信息，以帮助IT和业务管理层区分IT操作的优先级，并最大化业务成果。通过实时给出关键业务过程的指示，它提供了业务可用性上端到端的可见性和这些潜在IT基础结构上的映射。

1.1 历史上软件测试和开发并行

6

就某些方面而言，软件测试和自动化测试工具的发展与传统的软件开发的过程基本类似。以下是软件开发的一个简要发展过程，并展示了软件测试过程是如何从以前的最佳实践中转化的。

最早的计算机在20世纪50年代被发明出来，Fortran是第一种1GL编程语言。在20世纪60年代晚期，提出了结构化编程的概念，使程序可以在以下3种简单结构下编写：简单顺序、if-then-else，和do while语句。还有其他一些如何使程序成为一个“好程序”的先决条件，比如必须只有一个入口和一个出口等。当时的焦点在于创建程序的过程。

在20世纪70年代，开发团体把重点放在了设计技术上。他们认识到，要想保障质量，只有结构化编程是不够的——一个程序必须在编写其代码前加以设计。如Yourdon、Myers和Constantine等人的结构化设计技术和混合设计技术十分流行，并被认为是最佳方法。此时的焦点仍然是面向

过程的。

结构化设计的哲学把系统分割和组织成多个片段。分割意味着将问题分成更小的子问题，以使每个子问题都能最终对应于系统的一个片段。相关部分的问题应放在系统的同一片段中；也就是说，属于一类的事情放在一起。非相关部分的问题应放在系统不相关的片段中，这样彼此没有关系的事物不放在一起。

20世纪80年代，人们发现结构化编程和软件设计技术仍然是不够的：必须首先确立对程序的需求以保证能够交付给客户合适的系统。当客户收到了他们原先需要的特定程序时，他们通常把聚焦点放在质量方面。

出现了许多需求技术，例如数据流图（DFD）。数据流图的重要部分是存储，存储表述了应用程序将数据存储在哪里。存储的概念鼓励实践者开发逻辑视图来表示数据。先前人们的焦点是与数据相关的数据的物理视图。随后人们创建了数据模型的概念，以数据的方式对一个真实世界进行简化描述，比如数据的逻辑视图。这种方法的组成部分包括实体、关系、势、参照完整性和规范化等。这些也造成了一个过程和数据谁先谁后的论战，一个先有鸡还是先有蛋的问题。一直到逻辑表示数据法出现以后，人们的焦点逐渐从过程转移到了数据库上。数据的逻辑视图的倡导者最初坚持认为数据分析的首要焦点是数据，其次才是过程。随着时间流逝，人们逐渐开始一致同意了在定义系统的需求时过程和数据应该联合起来考虑的论点。

在20世纪80年代中期，引入了信息工程的概念。这是一个全新的学科，引领整个世界进入了信息时代。使用这种方法，在理解信息是如何存储和表示，信息如何以多媒体形式通过网络传输，以及不同的服务与应用程序如何处理信息等方面都是人们关注的热点。在数学和其他相关理论的帮助下，分析性的问题解决技术被应用到工程设计问题中去。信息工程强调了以企业的观点分配应用程序开发任务的重要性，而不仅仅是特定应用程序的观点。通过对所有企业就过程、数据、风险、关键成功因素和其他的维度建模，管理者就能知道能够有一个更有效的方法管理企业。

7

在同一段时间内，第四代计算机使用了微处理器芯片技术和速度惊人、存储量巨大的先进辅助存储设备。软件开发技术在飞速变化，第四代编程语言使得开发过程更加容易、快速。不幸的是，对于应用程序的快速开发周期的强调导致了基础开发技术有向尽快地“产出代码”倒退的趋势。这降低了对需求和设计的强调，这种情况目前在很多软件开发组织中仍在持续。

1.2 极限编程

极限编程（XP）是这样的倒退趋势的一个例子。极限编程是一种软件开发的非正统方法，由于缺少设计步骤而备受争议。极限编程方法倡导一种激进的、背离广泛接受的软件开发过程的方法。实际上有两条极限编程法则：尽量少的设计和没有需求分析而只有用户故事。极限编程原理坚持：“实际上没有法则，只有建议。极限编程方法每天或每周只需要用十分钟到半小时的时间做一个小的单元设计，周期性地完成。有效率的做法是，直到开始编程才开始设计。”

尽管在软件开发行业中大部分人认为需求文档是至关重要的，但是极限编程却建议尽量少地创建文档。极限编程中不预先创建需求文档，在软件开发过程中创建的文档也是非常少的。

使用极限编程方法，开发人员在做任何事情之前就需要先提出测试场景。测试先行的设计方

法要求开发人员在真实类开始编写之前就要首先编写好测试类,这样真实类的最终目的就不仅仅是满足需求,还要通过所有测试类中的测试。这种方法的问题在于需要进行独立的测试以发现产品中开发人员没有考虑到的问题或开发人员在他自己的测试中没有发现的问题。

1.3 自动化测试工具的发展

在20世纪80年代中期,随着自动捕获/回放工具的出现,测试自动化开始了。捕获/回放工具能帮助测试人员记录下交互场景。这些工具能记录下来每一个按键动作、鼠标移动和在这一场景中发送到屏幕上的回应。捕获/回放工具能自动记录下与预想结果的任何差异。这些工具减少了手工测试的工作量,大大提高了测试效率和生产力。

测试自动化的成本判断是十分简单的,可以用一个简单的图来说明。如图1-2所示,随着时间流逝,软件的业务操作的不断改变和改进,应用程序的功能特性数目不断增加。但是,测试每个新版本所投资的人数和时间总数是持平的,甚至还可能是减少的。因此,测试覆盖率逐步降低,这使得故障风险不断增加,甚至转变为潜在的业务损失。

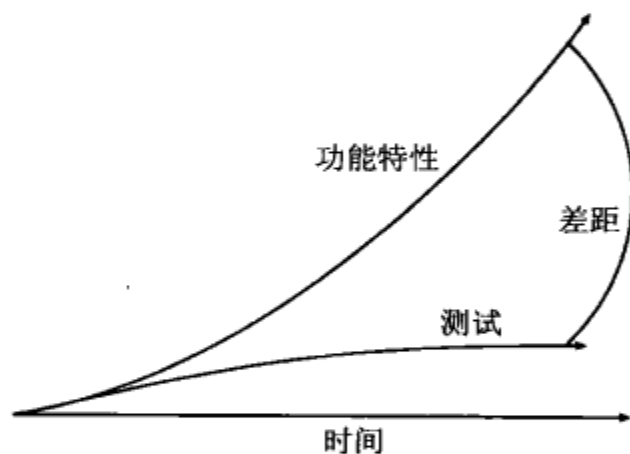


图1-2 测试自动化的动力(摘自“为什么要自动化”, Linda Hayes, Worksoft, Inc白皮书, 2002, www.worksoft.com, 授权使用)

例如,如果开发组织改进的应用程序中加强部分是现有代码的10%,这意味着测试工作量是以前的110%。由于没有任何组织肯为测试做出比开发更多的时间与资源的预算,测试者是很难跟上这一进度的。

这就是为什么投产多年的应用程序还经常出现故障。当测试资源和时间跟不上的时候,必须忽略对一些功能特性的测试。一般来说,最新特性会是测试重点,因为大家假设原来的特性仍能运行。然而,由于程序某个部分的改变经常对其他部分产生意料不到的结果(当然这一假设并不总是成立的),所以讽刺的事情发生了,最大的风险存在于已有的特性内,而非新特性上,因为一个很简单的原因:用户正在使用这些已有的特性。

测试自动化是解决这一两难问题的唯一方法。通过不断地在已有特性的自动化测试库中为新特性再加入新测试,测试库可以追踪整个应用程序的功能。

故障成本也在不断上涨。尽管在过去的几十年中,软件主要用于后台管理系统中,但今天软件是将许多公司与他们的竞争者区分开的极具竞争性的武器,并且这些软件逐渐组建了关键操作

的骨干系统。由于未检测出的软件缺陷而损失数千万或数亿甚至数十亿美元的例子数不胜数。周期的缩短也使得开发风险越来越大。产品周期从数年被压缩至数月、数周甚至数天。在如此紧张的时间限制中，想通过手工测试获得满意的测试覆盖率实际上是不可能的。

自动捕获/回放工具已经经历了一系列的阶段性改进。进化的改进描述如下。

1.3.1 静态捕获/回放工具（不附带脚本语言）

使用这些早期工具，测试是人工进行的，从后台捕获输入和输出。在接下来的自动回放中，脚本重复同样次序的操作来应用这些输入，并将实际回应与捕获结果相比较。一旦发现异样的结果，那么就会被报告为缺陷。GUI菜单、单选按钮、列表框和文本都可以存储在脚本中。使用这种方法限制了改变GUI菜单的灵活性。由这种方法中得到的脚本包括硬编码值，如果应用程序中有什么改变，这些值必须相应改变。维护这些脚本的成本简直是天文数字，完全无法接受。而且这些脚本几乎是不可靠的，甚至当应用程序没有改变时都会经常出故障（可能出现弹出窗口、消息和其他没有在录制测试时发生的事件）。如果测试者输入了一个错误的的数据，测试就需要重新录制。如果应用程序改变了，测试也要重新录制。

1.3.2 静态捕获/回放工具（具有脚本语言）

自动化测试工具的下一代引入了脚本语言。现在测试脚本已经是一个程序了。可以用脚本语言处理条件、异常和软件日益增长的复杂性。为了提高效率而开发自动脚本要受制于适用于软件开发的规则和标准。有效地使用任何自动化测试工具都需要至少一名受过培训的技术人员，即程序员。

1.3.3 可变的捕获/回放工具

新一代的自动化测试工具引入了附加的可变测试数据，与捕获/回放特性协同使用。静态捕获/回放和可变捕获/回放之间的不同是，前者的测试用例中输入和输出是固定的，而后的输入和输出是可变的。实现方法是手动运行测试，然后用变量对捕获的输入和预期的输出进行代换，这些变量对应值被存储在脚本外的数据文件中。使用一种具有可变数据能力的脚本语言使得大多数测试工具中的可变捕获/回放变得更有价值。可变捕获/回放及其扩展方法减少了对现有特性不进行回归测试的风险，并提高了测试过程的生产力。

然而，可变捕获/回放工具的问题是，它们仍然需要可以进行编程的脚本语言。但是随着开发程序技术的提高，也出现了新的脚本技术。

下面描述4种新技术。

- 数据驱动：数据驱动方法是从数据文件（如CVS文件、Excel文件、文本文件等）中读取测试输入和输出值来驱动测试的方法。

这种测试方法用可变数据重新强调了处理过程和数据的重要性，如1.1节中讨论的那样。必须聚焦在测试脚本与测试数据自动化，即开发数据建模。但是测试自动数据的创建常常是一种挑战。根据需求（如果有）创建测试数据是手工的、凭直觉的过程。将来的工具（如Smartware科技公司的SmartTest，一种测试数据生成工具）可以解决这个问题，办

法是用科学的方法生成可以导出到自动化测试工具的智能测试数据，作为可变数据，更多细节参见第34章。

- 模块化：模块化方法需要创建小的、独立的自动脚本和函数，这些脚本和函数代表正在测试的应用程序的模块、部分和函数。
- 关键字：关键字驱动方法是不同的界面、不同的函数和不同的业务组件被指定为数据表中的关键字的一种方法，测试数据和要完成的动作由自动化测试工具来生成脚本。
- 混和：混和是前面提到的所有技术的组合，并取其精华，去其糟粕。它是由核心数据引擎、通用组件函数和函数库定义的。然而函数库提供的通用例程非常有用，甚至在关键字驱动框架的上下文之外也是如此。核心引擎和组件函数高度依赖于所有这三种元素的存在。

11

每种技术的细节参见第28章。

质量保证框架

2.1 什么是质量

在韦氏字典中，质量是这样定义的：“质量是事物的一种本质特征，是与生俱来的，它有别于事物的其他特征，能够表明事物的优秀程度或级别。”如果查阅一下有关计算机的文献，你就会发现有两种关于质量的定义基本上可以接受。第一种将质量定义为“满足需求”，在这种定义之下，要有一个高质量的产品，需求必须是可度量的，这样才能知道产品的需求是否被满足了。也就是说，质量处于一种非此即彼的状态，就一个产品而言，要么是有质量的，要么就是没质量的。需求可以非常复杂，也可以非常简单，但只要这些需求是可度量的，就可以确定产品是否达到了质量要求。这是生产者的质量观，即用质量来表示产品是否满足了需求或达到了规约。从本质上讲，最终应该满足规约。

我们通常使用的是质量的另一种定义，它是从客户角度出发的。按照这种定义，客户把质量定义为产品或服务是否满足了客户的需求，换一种说法就是“适用”。通常在客户的“需求规约”（见附录C）里会描述产品的用途。需求规约是最重要的文档，质量体系的问题都要围绕它来解决。另外，在客户的需求规约中还要描述质量的属性，例如对易用性、可移植性和可复用性的描述。易用性是指用户和软件应用程序交互时的容易程度；可移植性是指软件系统在不同的硬件平台上执行时的兼容性；可复用性是指把为一个软件系统构建的组件复用在另一个软件系统中的能力。

13

每个人都应该对质量负责。然而，下面列举的许多人所持有的混淆不清的看法，实际上阻碍了对质量的追求。

- 质量需要承诺，尤其是来自最高管理层的承诺。为了实现高质量，要求管理层与员工紧密合作。
- 许多人认为无缺陷的产品和服务是不可能的，产品有一定的缺陷是正常的，是可以接受的。
- 人们经常把质量与成本联系在一起，即高质量就意味着高成本，这样在设计质量和构造质量上就出现了混淆。
- 质量要求需求规约足够详细，这样生产出的产品才能对照规约从数量上进行检测。但是许多组织都没有能力或不愿意花很多精力去写足够详细的规约。
- 技术人员常常认为标准会抑制他们的创造性，因此不愿意遵从标准的规定。但是，为了

保证产品质量，他们必须遵守具有明确定义的标准和规程。

2.2 预防与检测

质量是不能通过评估已经生产出来的产品获得的，因此，目标应该是首先去预防产品缺陷的产生，并且使产品可以通过质量保证度量进行评估。质量保证度量包括运用软件开发标准将开发过程结构化，并在开发过程中运用方法、技术和工具。软件中没有检测出来的缺陷可能会带来几百万元的损失，因此有必要发展独立测试，并且应该由公司来执行，而不是由软件系统的开发人员来执行。

就质量管理程序而言，不仅有必要进行产品评估，过程评估也同样非常必要，例如编码标准的文档化，标准、方法和工具的规定和使用，数据备份的规程，测试方法，变更管理，缺陷记录及修正等。

质量管理可以降低产品的成本，因为越早发现缺陷并加以修正，就越能减少最终带来的成本。随着自动化测试工具的出现，虽然最初要做一些投资，但从长远来看，将会获得高质量的产品，并能降低维护成本。

有效质量管理的全部成本是以下4部分成本的总和：预防、检测、内部故障和外部故障。预防成本是由最初为防止缺陷出现而采取的一系列行动产生的。检测成本是由测量、评估、审计带来的，其目的是使产品或服务符合标准和规范。内部故障成本是解决发货前发现的缺陷所发生的成本。外部故障成本则是解决产品发布以后才发现的缺陷所发生的成本。后者可能是破坏性的，因为这样会损坏公司的声誉，并对将来销售不利。

要想最大限度地获得回报，就要采取预防措施。重视预防成本就能降低产品到客户那里时的缺陷数，从而提高产品的质量，降低生产和维护的成本。

2.3 验证与确认

验证贯穿在整个开发生命周期中，用来评价产品是否满足了在前面一些已经正确完成的活动中的需求，而确认发生在生命周期的末尾，用来检查系统是否满足了客户的需求。已经证明，如果产品满足了用户的期望值，就能确保可执行系统如规约描述的那样运行。与验证相比，测试产品的创建与确认的关系更加紧密。从传统意义上讲，软件测试一直被认为是一个确认过程，也就是说，被当作生命周期的一个阶段。在程序编写完成后，对系统进行确认或测试，以确定其功能性和可操作性。

如果把验证整合到测试过程中，测试将贯穿产品开发的整个生命周期。实践证明，为了获得最好的结果，在测试过程中可以把验证与确认结合在一起。在软件开发生命周期中，验证包括评审、分析和测试的系统化过程，从软件需求阶段一直到代码编写阶段。验证保证了软件开发与维护的质量。同时，验证使得开发过程变得更加有组织、更加系统化，从而使开发出的应用程序很容易被第三方理解和评估。

验证是在20多年前为满足航空工业系统软件极高的可靠性要求的背景下出现的，因为程序中的一个错误就可能导致整个任务失败，造成巨大的时间和经济损失，甚至威胁到生命安全。验证

的概念包括两个基本标准：软件必须能适当地并正确地执行预期的功能，并且，软件一定不能自行执行某个功能或某些功能的组合，那样会降低整个系统的性能。验证的总目标就是要保证在软件生命周期中开发出的每个软件产品都能满足软件需求文档中描述的客户需求和目标。

验证还在软件文档各章节相关部分的需求规约之间建立了联系。努力做好全面的验证工作能够确保对规约中提到的所有软件性能和质量需求进行充分的测试，并且在发生变更后这些测试结果可以重现。验证是一个“持续改进过程”，没有明确的终点。为了维护配置和操作完整性，应该在系统的整个生命周期中都使用验证。

验证确保软件能实现预期的功能及所需的特性（如可移植性），增加了使软件只包含个别错误的可能性（即最终的产品中的错误数是可接受的）。验证为紧密监控软件开发项目提供了一个有效的方法，也为随时了解项目详细状态提供了管理手段。一旦应用了验证程序，就可以保证开发人员遵循一种正式的、有序的、可追溯的软件开发过程，从而以最少的代价获得系统质量的提高。

但对于验证，也有一种批判的观点，认为它会相当程度地带来软件开发成本的增加。然而，如果考虑从一开始到最后系统废弃的整个生命周期中软件开发的成本，实际上，验证能够减少软件的总体成本。如果能有效地运用验证程序，通常在安装系统中能把缺陷数减少1/4。因为在运行和维护阶段修正错误所花的成本大约是设计阶段的20~100倍，所以总共节省下来的成本要远远超出最开始对验证的额外投入。

2.4 软件质量保证

软件质量保证的正式定义是指为整个软件产品的适用性提供证据的系统化活动。软件质量保证是通过运用既定的质量控制方针来实现的，从而保证软件的完整性和更长的生命周期。事实上，质量保证、质量控制、审计功能与软件测试之间的关系经常被混淆。

为了生产出满足需求并且适用的软件产品，需要建立一系列过程并进行持续改进，质量保证就是为这些过程提供足够信心所需要的一系列支持活动。质量控制是把产品质量与可应用的标准相比较的过程，以及当检测出不一致时要采取的措施。审计是指验证产品是否符合计划、政策和过程的检查/评估活动。

软件质量保证是一项有计划的工作，以确保软件产品不仅符合标准，而且要具备本软件项目特有的属性，如可移植性、高效率、可复用性及灵活性。软件质量保证由一系列活动和方法组成，这些活动和方法用来监督和控制软件项目在预期的水准上实现特定的目标，它不仅仅是软件质量保证团队的责任，而且取决于项目经理、项目领导人员、项目全体成员以及用户的意见。

质量保证是负责管理质量的方法。保证的意思是指，如果遵循了一定的过程，管理层就可放心产品质量。质量保证具有催化功能，它能激励管理层和普通员工对待质量有端正的态度和严明的纪律。凡是成功的质量保证管理人员都知道怎样去培养员工的质量意识，并让他们认识到质量对他们自己以及整个组织的益处。

如果按照软件质量保证计划执行，软件质量的目标就一定能达到，计划中阐明的方法将被应用到项目中，以确保在每一个里程碑产出并评审的文档或产品都是高质量的。这样一种明晰的方

法能够保证为达到软件质量的要求,所有的步骤都被执行,同时向管理层提供了这些活动的文档。质量保证计划中阐明了一系列标准,依据这些标准,质量活动是可以被监督的,而不是设定一些根本不可能实现的目标,如软件没有缺陷或百分之百可靠等。

软件质量保证是一种风险管理的策略,它之所以存在是因为软件质量问题确实会造成成本的增加,因此应该纳入到项目的正式风险管理中。下面这些例子中的软件质量就很糟糕。

- 已经交付的软件频繁地出现故障。
- 系统运行失败所带来的后果令人无法接受,诸如经济损失甚至威胁生命安全。
- 系统经常不能实现预期的目的。
- 系统升级的成本经常很高。
- 发现并排除缺陷的成本过高。

大部分质量风险都与缺陷有关,但还不仅仅如此。缺陷的产生是由于不能满足某一项需求。如果需求本身是不恰当的,甚至是不正确的,那么缺陷的风险就会更大,结果就会产生很多内置的缺陷,而这些缺陷是根本无法验证的。在有些风险管理策略和技术中就包括了软件测试、技术评审、同行评审及兼容验证等。

2.5 质量保证的组成

大部分软件质量保证活动可划分为软件测试(即验证与确认)、软件配置管理和质量控制,但是软件质量保证程序的成功还同时依赖于标准、规程、惯例以及规约的统一构成,如图2-1所示。

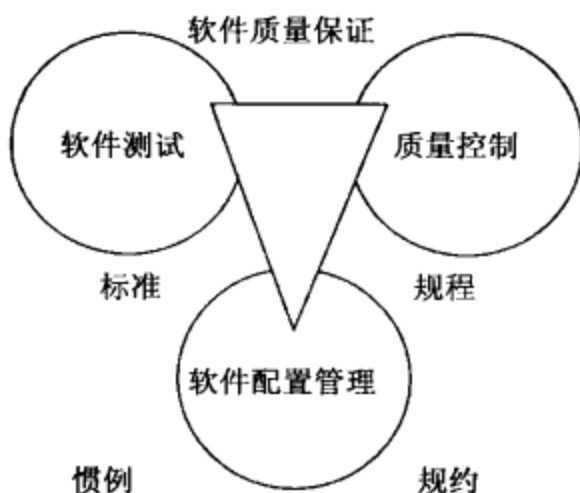


图2-1 质量保证的组成

2.5.1 软件测试

软件测试是一种很流行的风险管理策略,它可以用来验证产品是否满足了功能需求。然而,这种方法的局限性在于测试开始进行的时间对于控制产品的质量而言往往太迟了。测试与测试用例几乎一样,但它是可以被检查的,以确保通过所有可能的输入及系统状态的组合将所有的需求都测试到。然而在测试中并不是能发现所有的缺陷。软件测试包括文中提到的各种活动,例如验证与确认活动。在许多组织中,这些活动以及对这些活动的监管都被写在软件质量保证职责的章程里。让软件设计和代码编写之外的人员参与到软件质量保证活动中来应该是制度上、组织上及

项目政策上要考虑的事情。

验证与确认的主要目的就是要保证软件设计、代码及文档能满足所有的需求。需求的例子很多，比如用户需求，由用户需求导出并且通过设计用来满足用户需求的规范，代码评审和检测的标准，在模块级、子系统级以及集成软件级的测试需求，当代码与硬件充分集成后的用户验收测试等。

在软件设计和实现的过程中，验证可以用来帮助确定软件开发生命周期中某个阶段的产品是否满足前一阶段确立的需求。在整个开发过程中执行验证活动，只需花费较少的时间，并且也不复杂。

2.5.2 质量控制

质量控制可以定义为用来监督工作进展和观测需求是否被满足的过程和方法，它主要侧重于在产品发布以前评审和排除缺陷。质量控制应该是产品生产单位的责任。有可能是生产产品的同一组人来执行质量控制，也有可能是在产品生产单位内部再重新组建一个质量控制小组或部门。

18

质量控制是由针对产品的一系列定义明确的检查组成的，这些检查在产品质量保证计划中都有详细说明。就软件产品而言，典型的质量控制一般包括规约评审、代码和文档检查以及用户交付物检查。通常，在生命周期的每个里程碑都会进行文档和产品的审查，目的就是要验证这些文档和产品是否符合软件质量保证计划中说明的标准。具体讲，这些标准通常是在需求规约、概要设计及详细设计文档以及测试计划中进行说明的。提交给用户的文档包括需求规约、设计文档、用户验收测试的结果、软件代码、用户指南、操作及维护手册。其他附加文档则在软件质量保证计划中声明。

质量控制可以由不同的群体来执行。对于小项目，与项目组成员同级别的组或部门的软件质量协调人员就可以来审查文档。而对于大项目，应该由配置控制委员会负责质量控制，该委员会可以由用户或用户代表、软件质量保证部门的成员及项目负责人组成。

审查是质量控制的传统功能，它通过一些独立的过程来评估产品是否符合已经声明的标准。同行和领域专家通过评审规约和实际产品来鉴别缺陷并提出改进建议。它们被用来检查软件项目是否符合书面的项目规则，在项目里程碑达到时，或者在项目负责人或软件质量保证人员在项目生命周期中认为必要的时候。审查可以用用来评估是否符合的详细检查表，也可以用根据文档来确定交付物是否齐全的简要检查表。有关审查目的和所发现缺陷的报告要递交到项目监管人员、项目负责人及项目成员手里，以便于采取下一步的行动。

有关审查的职责是在软件质量保证计划中进行阐述的。对于小项目，项目负责人或部门的质量协调人员就可以执行审查。而对于大项目，应该由软件质量保证组的成员领导一个审计小组来执行审查，这个小组与前面提到过的配置控制委员会非常类似。审查完成后，就会分配项目成员按照明确的日程表去修正问题。

质量控制是用来检查和修正缺陷的，而质量保证是用来预防缺陷产生的。检测就意味着在期望生产无缺陷产品和服务的过程中可能会出现缺陷。质量保证是一种管理手段，它通过阻止缺陷、提出限制和重定向来达到预防缺陷产生的目的。

2.5.3 软件配置管理

19

软件配置管理关心的是在一个系统的软件部分进行贴标签、追踪以及控制其变更，通过管理软件组件的版本及各版本之间的关系来控制软件系统的演进。

软件配置管理的目的是要确定软件中所有相关的组件，并在不同的生命周期阶段来控制它们的演进过程。软件配置管理是一个规则，可以用于各种活动，包括软件开发、文档控制、问题追踪、变更控制及维护等，它在软件可复用性方面可以节省很多成本，因为每个软件组件与其他软件组件之间的关系都已定义清楚。

软件配置管理是由一系列活动组成的，这些活动保证了软件设计和代码能够被明确地定义，并且在没有对变更本身的影响及其对文档的影响进行评审前是不能进行变更的。配置管理的目的是控制代码及相关文档，以保证最后的代码及其描述与那些已实际评审并测试过的代码及其描述是一致的。这样，就消除了发布前的突发软件变更。

对于并行的软件开发项目而言，软件配置管理将带来相当大的益处。它可以在开发阶段就对软件进行组织，将由于疏忽造成变更的可能性降到最低。当遇到大量的变更活动时，或当面临选择了错误的软件组件而带来的巨大风险时，软件配置管理将会对所有软件确实实地产生影响。

软件配置管理的要素

软件配置管理之所以要确定系统的配置，是为了能够系统地控制变更、维护完整性、加强配置在整个生命周期中的可追溯性。需要控制的组件包括计划、分析及设计文档、源代码、可执行代码、实用工具、作业控制语言（Job Control Language, JCL）、测试计划、测试脚本、测试用例及开发报告等。典型的软件配置过程由4个要素构成，即组件确定、版本控制、配置构建、变更控制，如图2-2所示。

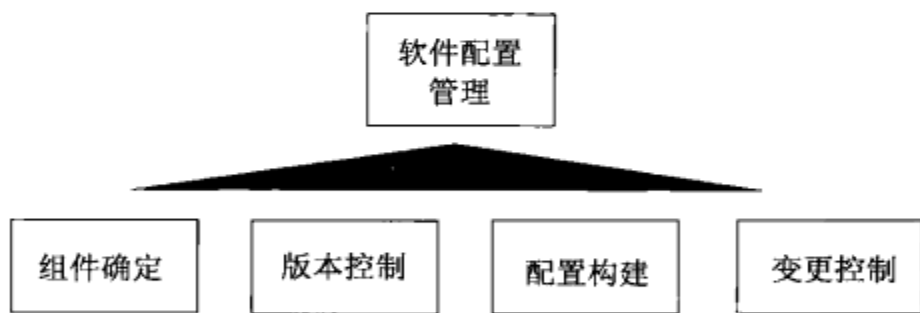


图2-2 软件配置管理

- 组件确定

软件配置管理的基本活动就是在开发过程中的每一个时间点确定软件的组件，以组成一个可交付物。软件配置管理提供了确定和命名软件基线、软件组件及软件配置的指导原则。

软件组件在开发过程中会经历一系列的变更。为了有效地管理软件开发过程，就必须建立一些方法和命名标准，以唯一确定软件的每一个版本。命名组件版本的一个简单方法就是采用离散的数字序列。第一个整数代表软件组件的外部版本号，第二个整数表示软件开发的内部版本号。

20

版本号从2.9转变到3.1就表明新的外部版本号3已经出现了。当组件被检入软件库时，软件组件的版本号会自动递增。如果有必要，还可以使用进一步的限定符，如新版本的日期等。

软件配置就是能完成一项主要业务功能的若干软件元素的集合，例如一个订单系统的一组程序模块。确定配置与确定单独的软件组件非常类似。配置可以有一系列的版本，每个配置都要以某种方式进行命名，以区别于其他配置。每个配置的版本都要与其他版本区别开来。一个配置的确定还必须包括其达到的状态，并描述清楚该配置是如何构建的。

确定软件配置的一个简单技术就是把所有软件组件都集中存放在一个存储库中，然后将所有组件的清单建档。

- 版本控制

随着软件应用的不断演进，会产生许多不同版本的软件组件，因此就需要一个有组织的过程来管理这些软件组件的变更及它们之间的关系；此外，通常还需要支持并行的组件开发及维护。

当软件通过一系列被称作版本的临时状态不断演进时，软件会频繁地发生变更。控制版本的软件配置管理机制实际上就是一个软件配置管理存储库。版本控制提供了每一次软件变更的历史和可追溯性，包括谁做了什么变更、为什么做及什么时候做的等。

在软件生命周期中，软件组件不断地演化，在某个特定的时间点，每个组件都会达到一个相对稳定的状态。但当修正缺陷或实现增强功能时，就会导致组件新版本的产生。维护这些软件组件版本的过程就称为版本处理。

对每个软件组件都要进行标识，以区别于该组件的所有其他版本。当修改软件组件时，应该对新旧版本分别加以标识。因此，除了最初的那个版本外，每个版本都有一个前驱版本。组件版本的连续性就代表了组件的历史和可追溯性。不同的版本也起到了备份的作用，使开发人员能够回到软件的早先版本。

21

- 配置构建

构建软件配置需要确定正确的组件版本，并执行组件构建程序，通常就把这一过程称为配置构建。

软件配置是由一组导出的软件组件构成的，例如可执行目标程序就是由源程序导出的。导出的软件组件要与每个源组件正确地关联起来，以获得准确的导出结果。配置构建模型确定了如何控制将导出的软件组件组合在一起。

配置构建模型需要的输入项主要包括源组件、版本选择过程、描述软件组件相互关系的系统模型等，输出则包括目标配置及其导出的软件组件。

软件配置管理环境使用不同的方法来选择版本，最简单的一种方法就是维护组件版本的清单，其他方法包括选择最新测试过的组件版本或在特殊日期修改过的组件版本。

- 变更控制

变更控制是修改软件组件的过程，包括修改建议的提出、评估、批准或否决、确定进度以及修改效果跟踪等活动，其基本过程包括变更控制过程、组件状态报告过程及审计过程等。

软件变更控制是用于控制软件变更的决策过程。在这个过程中提出的变更有些被接受，并进行了实施；而有些则被拒绝或延期了，没有实施。变更控制还为确定相互之间的依赖关系做好了影响分析的准备。

修改一个配置至少包括4个要素：变更请求、变更的影响分析、要修改的组件及要增加的新组件、将所做修改可靠地设置成新基线的方法（详见附录D）。

一项变更常常会涉及多个软件组件的修改。因此，用一个存储系统来存储某个文件的多个版本通常是不够的，于是就需要一项技术来确定发生一次变更时需修改的组件集合，这就是常说的delta存储。

22

每个软件组件都有它的开发生命周期。生命周期是由若干状态及状态之间允许的转换组成的。如果要修改一个软件组件，应该在新版本发布前对其进行评审，禁止（或冻结）其他修改。评审委员会必须选择批准或否决已修改的软件组件。软件存储库中存放的所有组件都是已经冻结的和已获批准的。

一个导出的组件会与源组件相关联，并与源组件具有相同的状态。而且，一个配置不可能比其中的任何一个组件具有更完整的状态，因为当某些相关组件没有冻结时，对配置进行评审是没有意义的。

软件配置管理控制的所有组件都存储在软件配置库中，包括工作产品如业务数据和过程模型、结构组、设计单元、被测应用软件、可复用的软件及专门的测试软件。当准备修改软件组件时，要将其从存储库中检出并存放到一个私有的工作区中，该组件可能会演进多个状态，并且会暂时处于配置管理控制的范围之外。

当变更完成后，软件组件又会被检入存储库，并生成一个新的组件版本。同时，前一个组件版本也会被保留。

2.6 软件质量保证计划

软件质量保证（SQA）计划是在软件开发中为保证质量水平所采取的有关质量控制手段的大纲。可以把这个计划当成基线，用来比较开发的实际质量水平与计划达到的质量水平。如果实际质量达不到计划质量水平，管理部门就会按照计划中描述的方法做出适当的反应。

质量保证计划为开发可理解的、可维护的代码提供了框架和方针，在软件项目中有助于保证质量的可追溯性。SQA计划还提供了一系列规程，以保证在公司内部或在合同项下开发或维护高质量的软件。这些规程会影响计划、设计、写代码、测试、写文档、存储及维护计算机软件。应该按照这种方式来组织这些规程，因为计划是用来保证软件质量的，而不是为开发和维护软件描述特定规程的。

开发和实施软件质量保证计划的步骤

1. 步骤1：编写计划

23

软件质量保证计划应当包括以下几部分（见附录B，其中有一个计划模板）。

- 目标部分——这部分主要描述特定SQA计划的具体目的和范围，它应该列出SQA计划所覆盖的软件条目的名称及软件预期的用途，还应该为每一个指明的软件条目说明软件生命周期中被SQA计划所覆盖的部分。
- 参考文档部分——这部分应该提供一个在SQA计划中所引用的其他地方的文档的完整清单。
- 管理部分——这部分描述项目的组织结构、任务及职责。

- 文档部分——这部分主要明确控制软件开发、验证与确认、软件使用及维护的文档，也说明了如何充分地检查这些文档，包括评审和审计的标准和验证过程，每个文档的充分性都要按照这一标准进行确认。
- 标准、实践、惯例及度量部分——这部分主要确定要应用的标准、实践、惯例及度量，并说明怎样监督和保证以符合这些条目。
- 评审及审查部分——这部分定义要执行的技术评审及管理评审、走查和审查，并说明怎样完成评审、走查和审查，包括后续的活动及审批。
- 软件配置管理部分——这部分会在项目的软件配置管理计划中详细描述。
- 问题报告和修正部分——这部分会在项目的软件配置管理计划中详细描述。
- 工具、技术和方法部分——这部分主要确定专门的软件工具、技术和方法，以支持SQA，并阐明它们的目的，描述它们的用途。
- 代码控制部分——这部分定义的方法和设施主要用来在开发过程的所有阶段维护、存储、保护、记录相应软件的控制版本，它可以与计算机程序库协作实施，或者可以作为软件配置管理计划的一部分。
- 介质控制部分——这部分主要描述了用来为每个计算机产品确定介质的方法，同时也描述了保存介质需要的文档，内容包括备份恢复过程，在开发过程的所有阶段保护计算机程序物理介质，以防止未经授权的访问或意外损坏或退化等。这部分可以根据软件配置管理计划编写。
- 供应商控制部分——这部分主要阐明为保证供应商提供的软件能够满足需求而制定的条款，而且也应该说明如何确保软件供应商能够收到充足、完整的需求。对于以前开发的软件，这部分阐明的方法主要用来确认SQA计划所覆盖软件条目是否适用于产品。而对于将要开发的软件，则需要供应商准备和实施SQA计划，并遵循本部分描述的条款。在这一部分中还会说明用来保证开发人员遵照本标准条款中需求的方法。
- 记录收集、维护、保存部分——这部分确定要保留的SQA文档，同时还会说明组合文档、保护文档、维护文档的方法和设施，并指明保留的周期。SQA计划的实施涉及必要的计划审批及计划执行的开展，并会进行后续的SQA计划评估作为执行的结果。
- 测试方法——这部分主要确定测试的方法、技术及用到的自动工具。

2. 步骤2：获得管理层认可

管理层的参与对于成功实施SQA计划是非常必要的。管理层既有责任保证软件项目的质量，又有责任提供软件开发所必需的资源。

实施SQA计划所需的管理层支持的级别取决于项目范围的大小。如果一个项目是跨组织的，就必须获得所有相关部门的批准。一旦获得了批准，SQA计划就会在配置控制下执行。

在管理审批流程中，管理层把严格控制软件质量的职责下放给SQA计划管理员，以换取软件质量的提高。软件质量常常会留给软件开发人员去保证。质量是令人渴望的，但管理层可能会更关心正式的SQA计划的成本。员工们应该意识到管理层评审程序是一种保证软件质量的手段，而不是目的。

按照管理层所关心的，应该正式地估计出项目实施中有SQA计划和没有SQA计划两种情况下

软件生命周期成本。通常，实施一个正式的SQA计划会带来经济和管理上的成本。

3. 步骤3：获得开发人员认可

因为软件开发和维护人员是SQA计划的主要使用人员，在计划的实施过程中他们的认可和协作是最基本的。软件项目团队的成员必须遵守项目的SQA计划，每个人都必须接受和服从它。

在计划的编制过程中，如果没有软件团队成员和他们的领导的参与，SQA计划就不可能成功实施。项目团队通常只有几个成员，所有成员都应该积极地参与到SQA计划的撰写中来。当项目变大时（如包括整个部门），就应该由子项目组的代表来提供输入。代表们给团队成员的不断反馈将有助于让计划被接受。

4. 步骤4：准备编写SQA计划

准备和起草SQA计划的过程需要人力资源和文档资源。负责实现SQA计划的人必须有权使用这些资源。而且，资源的交付使用需要管理层批准和支持。为促进资源分配，管理层应该清楚任何可能阻碍实施过程的项目风险（如人员或设备有限），并且应该制定草拟、评审、批准SQA计划的日程表。

5. 步骤5：执行SQA计划

软件开发和维护团队在执行SQA计划的实际过程中要确定用于监督的审计时间点。在软件产品的实现阶段必须将审计功能排入日程表，这样不适当的软件项目监督才不会影响SQA计划。审计点应该在开发过程中周期性地出现，或是在一些特殊的项目里程碑（如在主要的评审会议或项目的一部分要交付时）出现。

2.7 质量标准

接下来介绍一下IT业界主要的质量标准。

2.7.1 萨班斯-奥克斯利法案

2002年的《萨班斯-奥克斯利法案》，也就是著名的《2002年公众公司会计改革和投资者保护法案》，通常称为《SOX法案》或《Sarbox法案》，是2002年7月30日开始执行的美国联邦法律，这部法案是为了应对大量大公司的财务丑闻，包括安然公司、泰科国际公司、Adelphia公司、Peregrine系统公司和世界通信公司的丑闻。

设计《萨班斯-奥克斯利法案》是为了在企业内部确保以下几点。

- 对欺诈、滥用和财务数据/交易损失有足够的控制。在很多公司里，这些控制大多是基于IT的。
- 对启动快速检查有一定的控制，只要有这种问题出现。
- 为了限制这种问题的影响，应该采取有效的措施。

不仅要控制到位，控制还必须有效，并且必须可以记录由控制引发的异议，并且要遵循审计轨迹采取适当的行动应对这些异议。这一要求给了IT部门新的压力，直到现在还有一些IT部门无法面对。

COBIT的ISACA子集确保与《萨班斯-奥克斯利法案》相关的关键IT层面都被测试。正如

ISACA研究中推荐的那样，顶层的COBIT控制和满足这些控制的战术方案清单如表2-1所示。

表2-1 顶层COBIT控制

级 别	控制目标	要落实什么
1	网络安全	更新防火墙，确保无线传输
2	病毒防御	更新防病毒应用软件、防间谍应用软件
3	备份	正规的、经过测试的备份规程
4	文件访问权限控制	基于角色的访问控制，最低权限
5	IT作为战略计划的一部分	支持业务目标的技术
6	IT连续性和恢复计划	基本灾难恢复计划（DRP）规程
7	ID和验证规程	复杂密码，密码经常变更政策
8	管理支持/大宗买进	CEO领导IT控制项目
9	风险评估程序	
10	基本的风险评估和自身审计	培训电子邮件、网络和密码的使用
11	数据输入控制	字段格式，定期数据范围测试

COBIT目标是专门为辅助有效的信息和IT管理设计的，特别强调IT管理。这些为管理层提供了一个实现支持核心业务流程的内部控制系统的框架。澄清职责范围以及从事公司信息系统的管理、使用、设计、开发、维护和操作工作的所有个人的勤劳程度。

表2-2意在将318项COBIT控制分解到各个活动区域，试图让任务更便于管理。这有助于理解需要专注的关键区域，从内部控制的引入到自动的解决方案等。

表2-2 按活动区域划分的COBIT控制

一般活动	COBIT控制	说 明
IT计划和管理	83	IT过程的顶层控制元素。大量关注策略和职责的制定以及这方面的管理和报告
人力资源	17	IT过程中员工角色的确定，以及员工调动期间业务连续性和安全性问题
安全性	29	执行一个紧凑安全计划时涉及的职责和任务的确定
系统监控和效用	81	日常工作基础上系统的可用性和操作，还包括第三方支持
变更管理	11	你需要与变更管理厂商谈论开发一个高效的变更管理策略，最初的软件解决方案与市场中的多数已经建立的变更管理解决方案中的一些集成
数据管理	27	包括授权、协议、错误处理、安全等
测试	70	包括正式质量保证策略的建立和执行，最初的软件专注所有这些领域

2.7.2 ISO9000

ISO9000是一个质量体系，由国际标准化组织（International Standard Organization, ISO）

在1987年提出的5份文档组成。ISO9000标准及证书常常与非国际标准的生产过程联系在一起。然而, 应用软件的开发组织能够从这些标准中获益, 如果必要的话, 还可以给他们自己定位, 以用于认证。因为缺少严格性和规则, 所有的ISO9000标准都是原则性的和解释性的。在欧洲和美国的硬件制造业中, ISO认证变得越来越重要。同时, 被要求通过ISO认证的软件供应商也在不断地增加。ISO9000是一套权威的质量标准, 但它代表的质量标准只是全面质量管理(Total Quality Management, TQM)计划的一部分。它由ISO9001、ISO9002及ISO9003组成, 提供了选择和实施质量保证标准的方针。

ISO9001是一个非常综合的标准, 它定义了证明供应商具备设计和交付高质量产品的能力所需的所有质量要素。ISO9002为供应商控制设计和开发活动提出了有关质量方面的事项。ISO9003说明供应商在检查和测试过程中发现和控制不合格产品的能力。ISO9004描述与ISO9001、ISO9002及ISO9003有关的质量标准, 并能够提供完整的质量检查表。

表2-3所示为ISO9000和同类的国际标准。

表2-3 同类ISO标准

国际标准	美国标准	欧洲标准	英国标准
ISO9000	ANSI/ASQA	EN29000	BS5750(Part 0.1)
ISO9001	ANSI/ASQC	EN29001	BS5750(Part 1)
ISO9002	ANSI/ASQC	EN29002	BS5750(Part 2)
ISO9003	ANSI/ASQC	EN29003	BS5750(Part 3)
ISO9004	ANSI/ASQC	EN29004	BS5750(Part 4)

2.7.3 能力成熟度模型

软件工程研究所能力成熟度模型(SEI-CMM)是用于评价软件机构的软件过程能力成熟度以及识别提高过程成熟度所要求的关键实践的模型。当软件机构加强了其软件过程能力的时候, 他们的成熟度的级别会随之上升。每达到一个成熟度级别, 都意味着软件过程中一个不同的组成部分, 从而带来软件机构过程能力的整体提高。软件的能力成熟度模型描述了构成软件成熟度基础的原理和实践方法, 其目的是要帮助软件机构按照从特别混乱到成熟有序的软件过程演进途径中提升其软件过程成熟度。

CMM分为5个成熟度级别(见图2-3)。

(1) 初始级(initial)。软件过程以无序为特征, 有时甚至是混乱的。几乎没有什么过程是经过定义的, 成功完全取决于个人努力和豪言壮语。

(2) 可重复级(repeatable)。软件机构已经建立了基本的项目管理过程, 可跟踪成本、进度和功能的实现。已经建立起必要的过程规范, 基于以前的经验, 类似的软件应用项目也能够取得成功。

(3) 定义级(defined)。软件过程中无论管理活动还是工程活动都已文档化、标准化, 并集成到了软件机构的标准软件过程中。所有项目都使用该机构经过批准的、专用的标准软件过程来开发和维护软件。

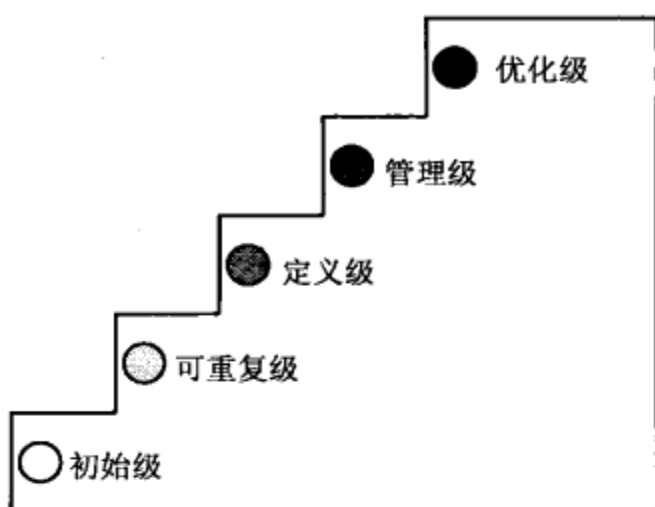


图2-3 成熟度级别

(4) 管理级 (managed)。软件机构收集了软件过程和产品质量的详细度量方法，从而可以定量地了解和控制软件过程和产品。

(5) 优化级 (optimizing)。通过过程的定量反馈和以试点方式采用创新思想及技术的定量反馈来驱动持续的过程改进。

1. 级别1：初始级

软件机构基本上没有为开发和维护软件提供一个稳定的环境。这一级别是很混乱的，软件机构没有为软件开发和测试建立任何规程和过程。当软件机构缺乏健全的管理实践时，无效的计划和无法兑现的承诺就会破坏优秀的软件工程实践带来的益处。

在这一级别，项目基本上摒弃了有计划的过程，回归到仅仅是写代码和测试上。成功完全取决于有一个出色的管理者和一支高效的软件队伍。项目的完成也依赖于有能力的项目经理。但当这些项目经理离开项目后，他们的稳定影响也随之离去。甚至很强的工程过程也无法克服由于缺少健全的管理实践而带来的不稳定性。

2. 级别2：可重复级

在这一级别有很多测量和度量需要评审，包括对各种不同过程的一致性的百分比、需求交付的百分比、需求变更的次数、项目计划变更的次数、估计的和实际交付的交付物之间的差异、PQA的实际审计结果与计划的差异，以及一段时间内处理的变更请求的次数等。下面是级别2中的主要过程活动。

- 软件配置管理。
- 软件质量保证。
- 软件转包合同管理。
- 软件项目跟踪和监督。
- 软件项目计划。
- 需求管理。

3. 级别3：定义级

在这一级别有许多测量和度量需要评审，包括花在测试活动上的时间占项目总时间的百分比、测试效率、交付物的检测率、检测效率、培训程序的实际出席人数与计划出席人数的差异，

以及管理人员的实际努力程度与计划的差异等。达到级别3意味着软件机构的管理过程和工程活动已经有了正式的定义,已经文档化,并且已经集成到了一个标准的过程中,软件机构的员工在开发和维护软件时能够充分理解和服从这一过程。软件机构一旦达到了这一级别,就具有了持续改进的基础。新的过程和工具能够以最小的不良影响引入进来,新员工经过培训很容易就能够适应软件机构的各项实践。下面是级别3的主要过程领域。

- 同行评审。
- 团体间的协作。
- 软件产品工程。
- 集成的软件管理。
- 培训程序。
- 组织过程定义。
- 组织过程重点。

31

达到级别3的软件机构的软件过程能力可以概括为标准的一致,因为无论是软件工程还是管理活动都是稳定的、可重复的。在已经建立的产品线中,成本、进度及功能都是可控的,并且软件质量也可以被跟踪。这样的过程能力主要基于整个组织范围内对于在预先定义好的软件过程中的活动、角色及职责的普遍认同。

4. 级别4: 管理级

这一级表示很好地定义了过程,并进行了专业化的管理。质量标准是不断提高的。有了健全的质量过程,软件机构就能更好地以合理的成本和尽责的交付物来满足客户对高质量/高性能软件的期望。交付一致的软件工作产品以及在软件开发生命周期内保证计划、过程、需求、设计、代码、测试的一致性将有助于拓展满意的客户群体。项目在运行过程中通过减少过程绩效的变化,使其尽量落在可接受的定量边界内,从而达到对产品和过程的控制。项目在运行过程中,特别是在已经建立的产品线中,可以将有意义的变动与随机的变动(噪音)区别开来。在新的软件应用领域提升学习曲线会包含很多风险,但这些风险是可以了解并加以仔细管理的。

- 软件质量管理。
- 定量过程管理。

达到级别4的软件机构的软件过程能力可概括为可预测,因为整个过程是可度量的,所有操作都处于可测量的限度内。这一过程能力的级别允许软件机构预测过程的发展趋势及定量限制边界内的产品质量。当超过这些限度时,就会采取措施来纠正。这样,软件产品就具有了可预见的高质量。

5. 级别5: 优化级

持续强调过程改进和减少缺陷将避免过程停滞或恶化,保证持续地改进并转化为生产力的提高、缺陷的减少及更加及时。在每个开发阶段跟踪需求可以提高软件的完整性、减少返工及简化维护。计划并执行验证与确认活动以减少缺陷。客户了解项目计划,定期收到项目状态报告,提供的反馈用于过程调优。级别5的KPA包括以下几项。

- 过程变更管理。
- 技术变更管理。

32

□ 缺陷预防。

达到级别5的软件机构的软件项目团队能够分析缺陷并确定错误原因。评估软件过程以防止已知类型的缺陷再次出现，并把取得的教训发布到其他项目中。达到级别5的软件机构的软件过程能力是以持续改进为特征的，因为达到级别5的软件机构总是在不断地努力提高他们的过程能力范围，因而能不断提高他们的项目的过程绩效。改进不仅体现在进一步提高现有过程，还体现在运用新技术和新方法进行创新上。

2.7.4 PCMM

人员能力成熟度模型（People Capability Maturity Model, PCMM）是帮助软件机构成功解决关键的人员问题的一个框架。基于当前人力资源、知识管理和组织发展等领域的最佳实践，PCMM可以给予软件机构一些指导，帮助他们在管理和发展员工方面提高自身的过程能力。PCMM帮助软件机构表现出员工实践成熟度的特征，建立员工持续发展的方案，设定改进行动的优先级，将员工发展与过程改进结合在一起，并打造出优秀的企业文化。PCMM自1995年发布以来，已经被全球数千家企业采纳，大小软件机构都有应用。

PCMM是由5个成熟度级别组成的，为持续提高员工个人的竞争力、发展高效的团队、激发绩效的提高及规范员工等建立了连续的基础，这些都是软件机构完成将来的业务计划所需要的。每个成熟度级别都是一个定义好的演化的水平，使发展员工新的能力成为制度。遵照成熟度的框架，软件机构能够避免引入那些员工没有准备好、不知如何进行有效实施的实践活动。

2.7.5 CMMI

CMMI产品套件为开发与维护产品和服务提供了最新、最好的实践方法。CMMI模型可作为开发和维护产品和服务的最好的过程改进模型。这些模型扩展了软件能力成熟度模型（SW-CMM®）、系统工程能力模型（SECM）及集成产品开发能力成熟度模型（IPD-CMM）中的最佳实践。

有软件机构报道说，尽管他们有许多专门的改进机会，但CMMI对于指导他们的过程改进活动非常充分，并且CMMI培训课程和评估方法非常适合他们的需求。对有些人来说CMMI的成本会影响他们是否采用的决策，但对其他人可能就不会。最后要强调的是，当软件机构决定是否在业务中采用CMMI时，投资回报率信息通常是非常有用的。

2.7.6 美国波多里奇国家质量奖

美国国家标准与技术研究院（NIST）如是说：

在20世纪80年代早期和中期，许多业界领导和政府官员看到重新兴起的对质量的重视不再仅仅是美国公司的选择，而成为在日益扩展的、需求更苛刻的、竞争激烈的世界市场上开展业务所必需的。但是许多美国企业并不相信质量对它们的重要性，或者根本不知道从哪里着手做起（Arthur, 1993）。

在1987年8月20日签署生效的100-107公法创立了波多里奇国家质量奖。评奖制度引发了新的

公私合作关系的创建。这一制度的主要支持来自于1988年建立的波多里奇国家质量奖的基金。该奖项是以Malcolm Baldrige先生的姓氏命名的，他从1981年开始担任商业部长，直到1987年在一次竞技比赛中意外去世。

波多里奇奖是美国总统颁发给企业（包括制造业和服务业、大规模和小规模）及教育和健康组织的，这些组织在以下7个领域公认是杰出的：领导能力、战略计划、客户及市场、信息与分析、人力资源、过程管理及商业结果等。……由于波多里奇奖和获奖者在美国质量运动中处于非常显著的中心位置，围绕着这个奖及其标准开展了更为广泛的国家质量行动方案。竞争委员会在一篇题为“21世纪的美国质量”的报告中讲到：“波多里奇质量奖比其他任何行动方案都更有责任使质量处于国家级的优先地位，并在全美推广最佳实践。”

每年大约有300多位来自工业界、教育研究机构、各级政府部门及非赢利组织的专家自愿花很多时间来评奖，进行实地考察，并为每一个申请者提供广泛深入的反馈报告。此外，委员会成员已经做了几千场关于质量管理、性能改进及波多里奇奖的演讲。

波多里奇奖的绩效优秀标准是一个任何组织都可以用来提高整体绩效的框架（见图2-4）。获奖标准是由7部分组成的（见表2-4）。

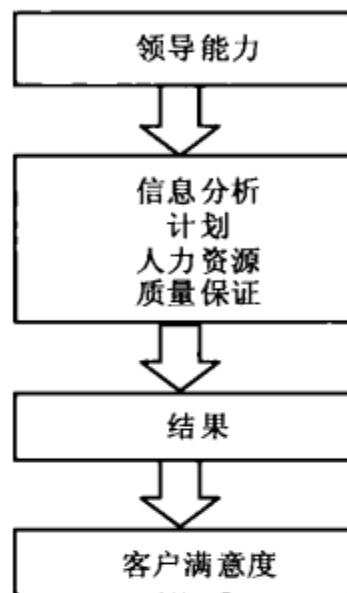


图2-4 波多里奇绩效框架

表2-4 波多里奇奖绩效框架

1. 领导能力——检查高级管理人员是怎样管理组织的，组织是怎样履行公共责任及很好地行使公民职责的。评估主要是基于适用性、有效性及领导人和公司与业务规模和类型有关的介入的范围。
2. 度量、分析及知识管理——检查数据和信息的管理、有效使用、分析及改进，以支持关键的组织过程及组织的绩效管理系统。数据的范围、管理及分析依赖于业务的类型、资源及地理分布。
3. 战略计划——检查组织怎样设定战略方向及怎样决定关键的行动计划。评估主要基于过程的完整性和有效性。
4. 人力资源——检查组织是怎样使员工充分发挥其潜质的，以及员工是怎样遵循组织的目标的。评估主要依赖于公司的人力资源管理方法。
5. 过程管理——检查主要产品/交付物及支持过程是怎样进行设计、管理和改进的。产品及服务的类型、客户需求及政府需求、调整的需求及经营地点的数量等因素都会影响过程管理。
6. 商业结果——检查组织在一些关键业务领域的运行和改进情况：客户满意度、财务及市场业绩、人力资源、供应商及合作伙伴的业绩、政府及社会责任等。同时，还检查组织怎样对待相关的竞争对手。
7. 客户及市场——检查组织怎样确定客户和市场的需求与期望，怎样建立与客户的关系，怎样获得客户，怎样使客户满意，以及怎样留住客户。

检查项的评分系统基于以下这些评价尺度。

(1) 方法——指公司用来达到目的的方法。方法应用的程度，工具、技术、方法的适用性及使用有效性，方法是否系统化、集成化及是否一致地应用，有效的自我评估及反馈，大量信息收集，以及方法的唯一性及创新性等，都是基于正确的方法进行决策的因素。

(2) 部署——这里主要关心方法的部署情况，将会评估方法是否在所有的产品和服务以及所有的内部过程、活动、设施及员工中进行了实施。

(3) 结果——这里主要关心方法的成果。已经证实的质量水平，质量改进的等级、范围、重要性，以及质量改进与已经证实的质量改进范围的比较等，都是影响结果的关键因素。

与其他程序（如ISO）相比较而言，日本的Deming奖和美国的波多里奇奖：

- 更注重结果和服务；
- 依赖于许多不同的专业队伍和行业队伍的参与；
- 能够为质量的创新方法提供专门的信誉；
- 强调关注客户及人力资源；
- 强调信息共享的重要性。

2.8 注释

1. <http://www.sei.cmu.edu/cmmi/adoption/cmmi-start.html>。
2. http://www.nist.gov/public_affairs/factsheet/baldfaqs.htm。

近10年来，软件测试作为一个独立的过程有了长足的发展，也得到了项目利益相关人和业务主管人员的高度重视。同时各种各样的新技术也在不断地引入到软件测试领域中。除了传统的测试技术之外，还有为适应当前复杂的业务需求和开发逻辑而引入的各种新技术，都使得软件测试变得更有意义，也更有目的性。本章讨论了一些被测试行业广泛采纳的、流行的测试技术。

3.1 黑盒测试（功能测试）

黑盒测试（又称功能测试）是一种测试条件以程序或系统的功能为基础生成的测试。也就是说，测试人员需要的信息是输入的数据和观察到的输出结果，但他们不知道程序或系统是怎样工作的。就像一个人不必知道汽车的内部是如何工作的就可以开车一样，我们不必知道程序的内部结构就可以执行它。测试人员侧重于根据规约去测试程序的功能。在黑盒测试中，测试人员把程序看做一个黑匣子，对程序或系统的内部结构并不关心。这一类测试包括决策表、等价类划分、范围测试、边界值测试、数据库完整性测试、因果图、正交表测试、数组和表测试、异常测试、极限测试、随机测试等。

黑盒测试的一个主要优点是测试活动本身的行为要与程序或系统的设计行为相吻合，这非常自然，所有人都可以理解。一些测试技术，如结构化走查、审查和JAD，可以证实这一点。黑盒测试的局限性是测试全部的、无遗漏的输入流是不太可能的，因为这要求每一个可能的输入条件或其组合都要被测试到。另外，因为测试人员不知道内部结构或处理逻辑，在黑盒测试中没有被测到的部分很可能会有致命的错误或程序员故意放置一段代码而搞的恶作剧。例如，假设一个为公司开发工资管理程序的程序员在自己开发的代码中嵌入一段“保护”自己工作的代码，如果这位员工被辞退，他的员工代码在系统中不再存在，而他预置在程序里的判断迟早会发挥作用。

```
if my employee ID exists
    deposit regular pay check into my bank account
else
    deposit an enormous amount of money into my bank account
    erase any possible financial audit trails
    erase this code
```

3.2 白盒测试（结构测试）

在白盒测试（又称结构测试）中，测试条件主要是针对检查逻辑路径来设计的。测试人员检查程序或系统的内部结构。测试数据根据对程序或系统的逻辑检查来驱动，而不去关心程序或系统的需求。测试人员知道程序的内部结构和处理逻辑，就像汽车维修工知道汽车的内部构造一样。这一类测试包括基本路径分析、语句覆盖、分支覆盖、条件覆盖、分支/条件覆盖等。

白盒测试的一个优点是测试比较彻底，并且侧重于已经开发出来的代码。因为测试人员知道内部结构或逻辑，一些致命的错误或程序员故意放置一段代码而搞的恶作剧就会很容易被检查出来。

白盒测试的一个缺点是不能验证规约的正确性，也就是说，白盒测试仅仅侧重于测试内部的处理逻辑，而不去验证逻辑是否满足需求规约。白盒测试的另一个缺点是无法检验代码中遗漏的路径和数据敏感性错误。例如，如果程序中的代码语句应该写成“if $|a-b| < 10$ ”，但却写成了“if $(a-b) < 1$ ”，没有详细的规约的话，这种错误将无法被检测出来。白盒测试的最后一个缺点是无法穷举程序中所有可能的逻辑路径，因为穷举导致的测试数量将会是一个天文数字。

3.3 灰盒测试（功能与结构测试）

黑盒测试主要是根据规约去测试程序的功能，白盒测试主要是测试程序的逻辑路径。灰盒测试是黑盒测试和白盒测试的有机结合。测试人员要研究需求规约并且与开发人员沟通以了解系统的内部结构，这样做的目的是清除一些有歧义的需求规约，掌握程序的逻辑以设计引申的测试。举例来说，当某一特定的功能在应用程序中会被重复使用的时候，测试人员可以采用灰盒测试。如果测试人员通过与开发人员交流并理解了内部的设计和架构，很多无效的测试就可以被排除掉，因为对于这个功能只测试一遍就行了。还可以举另一个例子：当命令行语法包含如下7个可以以任何顺序输入的可能的参数时，理论上，测试人员将不得不进行7!次（即5040次）不同的测试。

```
Command parm1, parm2, parm3, parm4, parm5, parm6, parm7
```

如果一些参数是可选的，在这种复合情况下问题会更加复杂。如果测试人员采用灰盒测试，通过与开发人员交流，理解了内部的语法分析算法，如果每一个参数都是独立的，那么仅用7次测试就可以满足要求了。

3.4 手工测试与自动化测试

手工测试的分类根据是该类测试不是由人在计算机上执行的。这类测试的例子包括结构化走查、审查、JAD和桌面检查。

自动化测试的分类根据是该类测试是在计算机上执行的，如边界值测试、分支覆盖测试、原型法和语法测试。语法测试是由一种语言编译器来实现的，而语言编译器是在计算机上执行的一种程序。

3.5 静态测试与动态测试

静态测试方法是与时间无关的，不需要被测软件产品的手工执行或自动执行。例子包括语法

41

检查、结构化走查和审查。程序的审查是对源代码清单中的代码一行一行地阅读并加以讨论。使用计算机的静态测试的例子是静态流分析工具，这种工具可以测试另一个程序中的错误而不必执行它。该工具分析其他程序的控制流和数据流来发现一些问题，如引用没有初始化的变量，或永远用不到的代码等。

动态测试技术具有时间依赖性，包含了在纸面上或在计算机上对一些指令的执行。例子包括结构化走查，在结构化走查过程中程序逻辑通过一步一步地执行代码而被模拟出来，并且使用口头描述加以配合。边界值测试也是动态测试技术，要求测试用例在计算机上执行，特别要关注与程序的输入和输出相关联的边界值。

3.6 软件测试技术的分类

42

测试技术是指顺利完成测试的一系列相关过程，有很多可能的分类方式，表3-1是其中的一种。表中列出了流行的测试技术，也按照上面的讨论对其进行了分类：手工测试、自动化测试、静态测试、动态测试、功能（黑盒）测试或者结构（白盒）测试等。

表3-2描述了每一种软件测试方法。

表3-1 测试技术分类

测试技术	手工测试	自动化测试	静态测试	动态测试	功能测试	结构测试
验收测试	×	×		×	×	
即兴测试	×				×	
α测试	×			×	×	
基本路径测试		×		×		×
β测试	×			×	×	
黑盒测试		×		×	×	
自底向上测试		×		×		×
边界值测试		×		×	×	
分支覆盖测试		×		×		×
分支/条件测试		×		×		×
因果图		×		×	×	
比较测试	×	×		×	×	×
兼容性测试	×	×				×
条件覆盖测试		×		×		×
CRUD测试		×		×	×	
数据库测试		×		×		×
决策表		×		×	×	
桌面检查	×			×		×
端到端测试	×	×			×	
等价类划分		×		×		
异常测试		×		×	×	
探索测试	×			×	×	

(续)

测试技术	手工测试	自动化测试	静态测试	动态测试	功能测试	结构测试
自由形式测试		×		×	×	
灰盒测试		×		×	×	×
直方图	×				×	
增量集成测试	×	×		×	×	
审查	×		×		×	×
集成测试	×	×		×	×	
JAD	×				×	×
负载测试	×	×		×		×
突变测试	×	×		×	×	
正交表测试	×		×		×	
帕累托分析法	×				×	
性能测试	×	×		×	×	×
正反测试		×		×	×	
缺陷历史预测	×		×		×	
原型法		×		×	×	
随机测试		×		×	×	
范围测试		×		×	×	
恢复性测试	×	×		×		×
回归测试				×	×	
基于风险的测试	×		×		×	
运行图	×		×		×	
三明治测试		×		×		×
健全性测试	×	×		×	×	
安全性测试	×	×				×
状态转换测试		×		×	×	
语句覆盖测试		×		×		×
统计概况测试	×		×		×	
压力测试	×	×		×		
结构化走查	×			×	×	×
语法测试		×	×	×	×	
系统测试	×	×		×	×	
表测试		×		×		×
线索测试		×		×		×
自顶向下测试		×		×	×	×
单元测试	×	×	×			×
易用性测试	×	×		×	×	
用户验收测试	×	×		×	×	
白盒测试		×		×		×

表3-2 测试技术描述

测试技术	简要描述
验收测试	基于最终用户/客户规约的最终测试, 或基于最终用户/客户使用一段时间后进行的测试
即兴测试	与探索测试相似, 但是通常指测试人员在测试以前对软件有较深的理解
α 测试	当开发接近结束的时候对应用程序进行的测试; 作为测试结果, 可能会有一些细微的设计变更。通常由最终用户或其他人员完成, 而不是开发人员和测试人员完成
基本路径测试	基于程序或系统的流和路径进行的测试
β 测试	当开发和测试基本上都结束的时候对应用程序进行的测试; 产品最终发布以前, bug或问题需要在该测试中发现。通常由最终用户或其他人员完成, 而不是开发人员和测试人员完成
黑盒测试	测试用例的产生是基于系统的功能的
自底向上测试	从下面开始对模块或程序进行集成测试
边界值测试	测试用例是由等价类的边界值产生的
分支覆盖测试	验证每一条分支取真取假各至少一次
分支/条件测试	验证每个判断以及判断的每个条件取所有可能结果至少一次
因果图	通过映射同时发生、互相影响的多个输入来确定要测试的判定条件
比较测试	与竞争对手的产品比较其优势与劣势
兼容性测试	测试软件与特定的硬件/软件/操作系统/网络等环境的兼容性
条件覆盖测试	验证一个判断的每个条件取所有可能的结果至少一次
CRUD测试	建立CRUD矩阵并测试所有目标的生成、读取、更新和删除
数据库测试	检查数据库字段值的完整性
决策表	显示决策标准和相应的行动的表
桌面检查	开发人员评审代码的正确性
端到端测试	与系统测试类似, 测试尺度的“宏观”端; 包括在一个模拟真实世界使用的情况下对完整的应用程序环境进行的测试, 这种模拟包括与数据库交互、使用网络通信或在适当的情况下与其他硬件、应用程序或系统的交互等
等价类划分	每一个输入条件都被划分入两个或多个分组。测试用例从有效类和无效类的代表值中生成
异常测试	识别出错消息和异常处理流程以及触发它们的条件
探索测试	经常被看做一个创造性的非正式的软件测试, 这一测试不是基于正式的测试计划或测试用例的, 测试者可能在测试软件的同时正在学习该软件
自由形式测试	利用直觉定义测试用例的即兴测试或以头脑风暴
灰盒测试	白盒测试和黑盒测试的组合方式, 充分利用了二者的优点
直方图	测量值的一个图形表示, 这些测量值根据出现频率分类组织, 用于定位热点
增量集成测试	当在一个应用程序中加入新功能时对其进行的继续测试, 需要一个应用程序功能的不同方面足够独立以能够在程序的所有部分完成之前单独工作, 或者测试驱动是按照需求进行开发的, 该测试由程序员或者测试人员完成
审查	同事之间正式的代码审核, 会使用到检查表、准入标准和退出标准
集成测试	对一个应用程序的各个组合部分的测试以确定它们的功能是否正确地结合。这些部分可以是代码模块、个体应用程序或者在一个网络上的客户/服务器应用程序。这种测试类型与客户/服务器结构的系统和分布式系统的关联尤其紧密
JAD	用户和开发人员坐在一起, 用易于理解的会话方式共同设计系统

(续)

测试技术	简要描述
负载测试	在很重负载的情况下对应用程序加以测试,例如,在一个负载的范围下对网站进行测试以确定在哪一点系统的反应时间会变慢或瘫痪
突变测试	决定一组测试数据或测试用例是否有用的方法,通过故意引入不同的代码变动(bug),并用原始测试数据/用例重新测试以确定是否这些bug能被检测出来。这一方法的适当实现需要大量的计算资源
正交表测试	确定哪些变量的改变需要被测试的数学技术
帕累托分析法	对缺陷模型加以分析以识别原因和来源
性能测试	可与压力测试和负载测试互换使用的方法。理想情况下,性能测试(以及其他任何测试类型)应在需求文档或者QA或测试计划中定义
正反测试	对所有输入测试正确值和错误值
前缺陷历史测试	为在系统的前一次测试中发现的每一个缺陷创建或者重运行测试用例
原型法	通过建立一个潜在应用程序的某些部分并向用户展示来从用户处收集数据的一般方法
随机测试	涉及从一个特定的输入值的集合(其中的值与其他值非常相似)中随机选择的技术
范围测试	对于每一个输入,找出系统反应相同的区间范围
恢复性测试	测试一个系统从崩溃、硬件故障或其他灾难性问题中能够恢复到什么程度
回归测试	根据在一个开发螺旋周期或者一个新版本的调试、维护或开发中产生的变化对应用程序加以测试
基于风险的测试	测量一个系统所具有的业务风险的程度以对测试加以改进
运行图	一个关于质量特性怎样随时间变化的图形表示
三明治测试	同时使用自顶向下和自底向上技术对模块和程序进行集成
健全性测试	一般来说是一个初始的测试工作,用以确定一个新的软件版本是否运行足够良好,达到一个可以进行主要测试的标准。例如,如果新软件每5分钟就系统崩溃一次、系统运行陷于停顿状态或者毁坏数据库,那么这个软件可能就处于不够健全的情况,无法在其现有状态下保证进一步测试的顺利进行
安全性测试	测试系统如何抵制未授权的内部或外部访问、故意损害等,可能需要复杂的测试技术
状态转换测试	首先标识了一个系统的状态,然后编写一个测试用例以测试造成从一个状态转换到另外一个状态的触发条件的技术
语句覆盖测试	确保程序中的每一条语句至少执行一次
统计概况测试	用来描述系统的使用概况的统计技术,有助于确定事务路径、条件、功能和数据表格
压力测试	可与性能测试和负载测试互换使用的方法。用于将这样的测试描述为在非正常的高负载、特定行为或输入的大量重复、输入大量数值数据或对数据库系统的大量复杂访问的情况下的系统功能测试
结构化走查	举行一个项目相关人员对工作产品进行查错的会议
语法测试	测试输入语法组合的数据驱动技术
系统测试	基于整体的需求规约的黑盒类型测试,覆盖了一个系统的所有组成部分
表测试	测试表项的访问、安全性和数据完整性
线序测试	将个体单元组合成为共同完成一个或一组功能的功能性线索
自顶向下测试	从顶部开始整合模块或程序

(续)

测试技术	简要描述
单元测试	测试的最微观尺度，测试特定的功能或代码模块，一般来说由开发人员而非测试人员进行，因为它需要对程序内部设计和代码有细致的了解。一般不容易实现，除非应用程序的代码具有非常好的结构。可能需要开发测试驱动模块或测试执行器
易用性测试	测试软件是否用户友好。很明显这是主观的，并且依赖于目标最终用户或客户。可使用用户访谈、调查、用户会议的视频记录和其他技术。开发人员和测试人员通常不适合作为易用性测试人员
用户验收测试	确定软件是否让最终用户或客户感到满意
白盒测试	通过检查系统的逻辑路径来确定测试用例

将需求转换成可测试的测试用例

4.1 概述

质量保证是一个过程，涉及整个开发和生产过程，包括监视、改进、找出并修复存在的问题和bug。

软件测试是软件开发生命周期的重要部分。有些组织将测试任务分派给测试程序员或者质量保证部门，有些组织则外包出去（参见第33章）。在软件测试过程中，质量保证团队通常由开发人员、测试人员和业务团队共同组成，他们互相协助、共享信息、彼此分派任务。

下面一节将概述有“良好”的需求时应如何创建测试用例。

4.2 将软件需求作为测试的基础

你会盖一所没有明确结构和特定需求的房子吗？当然不会，因为这只会白白浪费原材料和劳动力。但是有这样一个普遍的共识：软件开发工作与此不同，它首先进行基本构建工作，宣告构建成功之后，再花大量的时间来修复和重建软件。这就是所谓的“维护”。根据斯坦迪什集团（Standish Group）的统计，美国的公司每年在失败的软件项目上花费84亿美元，在显著超出时间预算和资金预算或功能下降的项目上花费138亿美元。

图4-1表明，如果整个项目的成本有8%~14%投入到需求活动中，那么项目成功的可能性是最大的

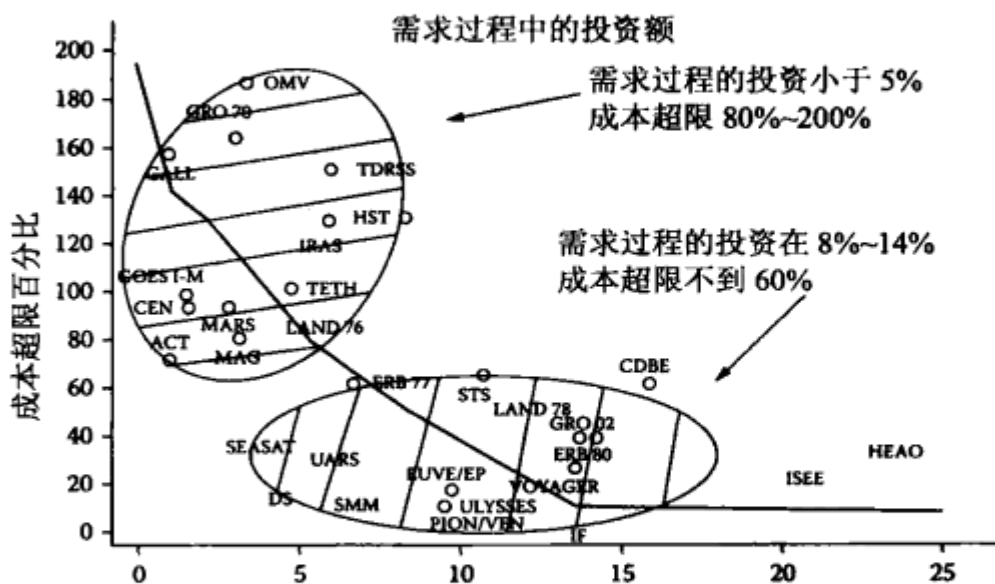


图4-1 良好需求的重要性（引自Ivy Hooks）

(充分满足目标成本要求)。

4.3 需求质量因素

如果软件测试取决于良好的需求，那么理解质量需求的一些关键因素就很重要。

4.3.1 可理解

需求必须是可理解的，可理解的需求的组织方式便于评审。下面是一些提高可理解性的方法。

- 根据对象（如客户、订单和发票）来组织需求。
- 用户需求应该按业务过程或场景进行组织，这样，行业专家就可以判断是否有遗漏的需求。
- 将功能需求与非功能需求分开，例如，将功能需求与性能需求分开处理。
- 根据详细程度来组织需求。这决定了需求对系统的影响，例如“系统应该能够接收订单”与“系统应该能够接收销售点的零售订单”对系统就有不同的影响。
- 编写出来的需求应该符合语法规则，样式应该便于评审。如果需求是用Word编写的，就应启用拼写检查选项，但同时也要注意上下文，因为拼写检查选项可能认可某个词或短语，但不注意上下文是否合适。
- 在需求中恰当使用“应该”(shall)一词，而不要使用“将会”(will)或“应该会”(should)，因为后面两个词讲的是目标，而不是需求。如果使用非命令式的词语，表明需求是可选的，可能提高成本，也可能使进度表加长，降低质量，或者导致合同纠纷。

52

4.3.2 必需

需求是必需的。现在举一个非必需需求的例子。假设下面这个需求已包含在需求规格说明中：“如果有100个测试用例通过，那么系统应该是可接受的。”这实际上是一个项目过程，而不是一个需求，不应该记在需求规格说明里。注意，需求必须与正在构建的目标应用或系统相关。

4.3.3 可修改

需求及相关信息必须是可修改的。用于存储需求的技术影响着可修改性，例如，字处理器中的需求比需求管理工具（如CaliberRM或Door）中的更难以改进。但是，对于非常小的项目而言，因为需求管理工具的成本和学习曲线的关系，字处理器就成了最佳选择。

一致性也对可修改性有影响。需求的编写模板和涉及的词汇表使得全局修改是可行的。选用的模板结构应该使需求可见，从而方便可修改性。好的最佳实践会把每个需求用一个唯一的标识符进行标记。需求应该位于显著位置，方便查找。任何需求依赖关系都应该有标注，例如，需求Y应该依赖于需求X。

4.3.4 非冗余

不应该出现重复的需求，否则会引发问题。重复需求会加重维护工作，也就是说，每次修改

某个需求时，也要修改与之重复的那个需求。重复需求还会提高出现注入需求错误的可能性。

4.3.5 简洁

一个好的需求必须杜绝出现冗词赘句或多余信息。一个简洁的需求表达应该一语中的，诸如“另一方面”、“然而”、“回想起来”这类字眼纯属假学者文风。

4.3.6 可测试

可测试需求必须可以被验证或确认，也就是说，需求的意图应该是能够证明的。不可测试的需求适合测试人员要做客观解释的时候。好的最佳实践是假装计算机不存在，然后问你：我是否可以测试这个需求，了解它能否行得通？

4.3.7 可跟踪

需求必须是可跟踪的，这是验证是否满足需求的关键。复合需求很难跟踪，可能导致产品测试失败。例如，“系统应该计算出退休金和遗属抚恤金”就是一个复合需求。通过列表方法可以避免在评审各个需求的跟踪能力时造成误解。

4.3.8 在范围内

所有需求必须在指定范围内定义。项目的范围是根据为项目建立的所有需求而定的。项目范围按照对需求的标识、分析和基准线来定义和细化。跟踪能力矩阵有助于保证需求位于指定范围内。

4.4 评估需求质量的数值方法

确保质量需求的一个最佳实践就是使用数值测量方法，而不是使用“能力差的”、“可接受”、“好的”、“完美的”等主观限定词。

这种方法的第一步就是创建一个用于需求评审的需求质量因素的检查表，第二步是根据重要性给每个质量因素加一个权重。所有因素的总权值是100。例如：

- 质量因素1 = 10;
- 质量因素2 = 5;
- 质量因素3 = 10;
- 质量因素4 = 5;
- 质量因素5 = 20;
- 质量因素6 = 15;
- 质量因素7 = 10;
- 质量因素8 = 25。

质量因素的总分值是100，不合适的质量因素要从总数中减掉。例如，如果除了质量因素5以外的其他质量因素都满足了，那么从100中减去20，结果得到最终的分值80%。

4.5 根据好的需求创建测试用例的过程

一项技术就是做某件事的过程、方式和方法。附录G描述了39种软件测试技术，引入的示例包括黑盒、白盒、等价类划分等。这些技术都用在方法论中。

一种方法论或一个过程就是指为其应用领域提供方法和原则的原理、指导原则或者蓝图。在信息系统环境中，方法论就是指重点关注收集信息、制定计划和设计元素的策略。

下面概述了一种从良好的需求扩展出测试用例的方法论。

4.5.1 步骤 1：评审需求

在编写测试用例之前，需要评审需求以确保它们能反映需求的质量因素。

审查是一种为发现许多问题而进行的正式、严格的团队手动同行评审，单个的评审人员都无法独自完成这项任务。非正式的同行评审也是可行的，这要看具体情况。然而，需求评审并不总是有效。（参见第6章，了解有关审查和其他评审的详情。）

下面是两种自动处理需求的流行工具。

- Smartware Technologies公司提供的商用SmartCheck™。这是一种自动化文档评审工具，用于根据词语、短语、字词范畴和复杂性来查找需求和技术规约中的不规范和歧义。它有一个词汇表，收纳的是容易引起歧义和有结构错误的词语。SmartCheck还可以让用户往字典里自行编辑和添加词语、短语和词语类别。SmartCheck报告会提交18种可能的不规范词语或短语的出现频率分布情况。这个工具不会有意评估指定需求的正确性，它是正确编写需求而非编写正确需求的一个有力助手。

下面的例子就体现了通过运行SmartCheck得到的结果，这段引语摘录自美国《独立宣言》。虽然这不是一个软件需求规约，但是却道出了规约的重点所在。

In every stage of these Oppressions We have Petitioned for Redress in the most humble terms: Our repeated Petitions have been answered only by repeated injury. A Prince, whose character is *thus* **<--a subordinate conjunction to connect ideas-consider rewording^①** marked by every act which *may* **<--a potentially ambiguous condition-consider rewording^②** define a Tyrant, is unfit to be *the ruler* **<--a potentially ambiguous noun or variable^③** of a free people.^④

图4-2中的SmartCheck™报告说明了在18种不规范基础上的词语或短语的分布情况，图4-3中的SmartCheck™报告说明了这18种不规范所属类型的分布情况。（欲了解详情，请参见网站<http://www.smartwaretechnologies.com/>。）

① 意思是：这是连接语义的从属连词，请考虑重述。——译者注

② 意思是：可能出现歧义条件，请考虑重述。——译者注

③ 意思是：这是可能出现歧义的名词或可变因数。——译者注

④ 这段英文的意思是：“在这些高压政策的每一个实施阶段，我们都曾以最谦卑的态度请求予以纠正，而一次次的呼吁所遭遇的回应却是一次次的迫害。一个一举一动都表现得像个暴君的君王，不配做自由民族的统治者。”

——译者注

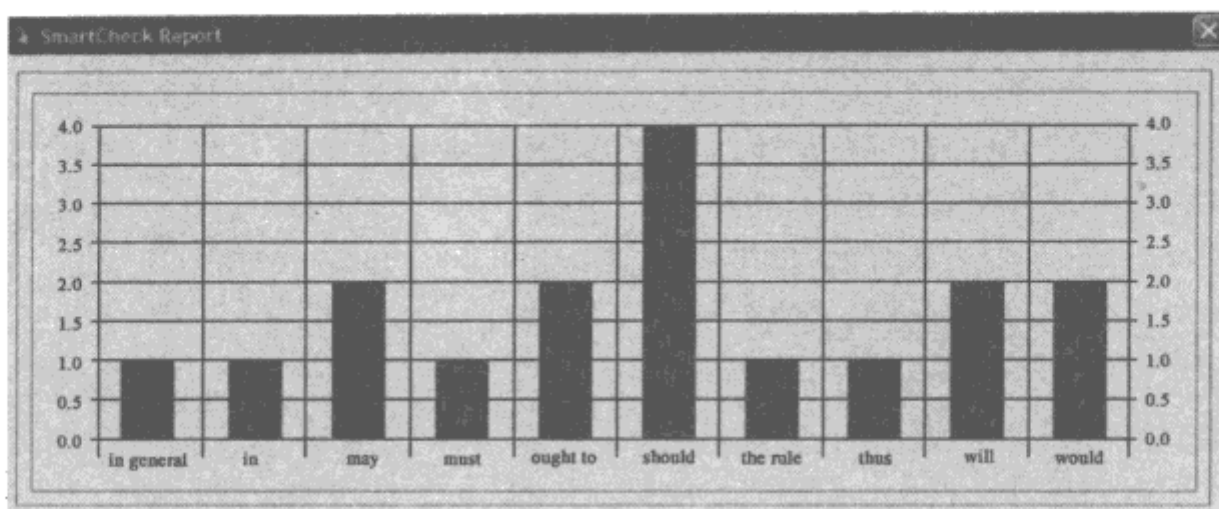


图4-2 SmartCheck™词语/短语分布报告

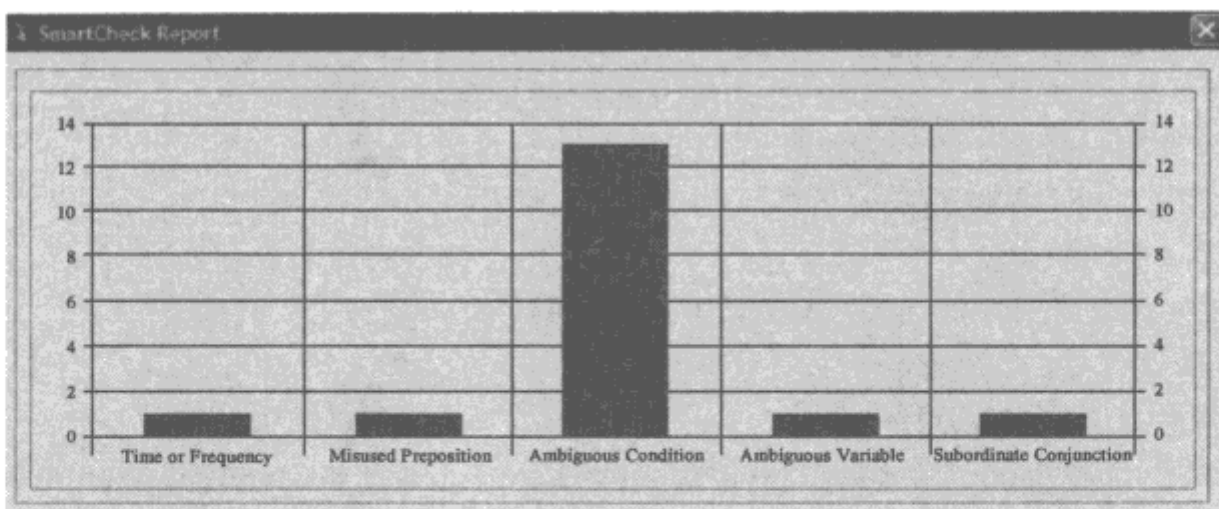


图4-3 SmartCheck™不规范类型的报告

- ARM工具（自动化需求测量工具）是NASA戈达德空间飞行中心的软件保证技术中心（Software Assurance Technology Center, SATC）开发的一种生命周期早期的测量工具，用于评估用自然语言表述的需求。ARM工具的目的是提供测量结果供项目经理来评估需求规约文档的质量，它扫描需求规约文档的关键字词和短语，生成一个报告文件，总结特定的质量指标。（欲了解详情，请参见网站<http://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php>。）

56

下面是一些用于改进测试过程的需求评审提示。

(1) 为评审人员做准备。在实际评审之前为评审人员提供需求说明，告诉他们你需要什么样的输入，以及如何研究和分析需求规约。例如，告诉他们你需要评审哪些部分。

给评审人员提供一份常见需求错误的检查表，使之明确检查重点。（参见本书附录提供的几个检查表。）

告诉评审人员应如何评审，确保参与者明白怎样才能高效而且有建设性地协作，还要告诉他们不要出现愚蠢的问题。

(2) 邀请合适的评审人员。选定需要参与需求评审的人员的类型，例如，开发人员、主题专家（SME）、业务分析师和测试人员。

(3) 强调找到主要问题的重要性。评审人员的真正作用在于找到主要的过失错误和遗漏错误。找到这种错误可以帮助避免在项目后期付出庞大的工作量和高昂的代价返工。

(4) 提出有用的问题。下面是一些需要在评审中提出的有用的问题。

- ☐ 软件产品有明确的目标和目的吗？
- ☐ 已确定产品用户或用户组的特征了吗？
- ☐ 已规定软件的所有外部接口了吗？
- ☐ 每个需求都有唯一的标识或标签吗？
- ☐ 已明确所有的条件了吗？
- ☐ 已指定所有的操作了吗？
- ☐ 已对需求进行逻辑分组了吗？
- ☐ 已对需求进行层次结构分组了吗？
- ☐ 已对需求进行优先级划分了吗？
- ☐ 定义了需求的类型吗？例如，是功能方面的还是性能方面的？
- ☐ 各个需求都一致吗？有冲突吗？
- ☐ 需求是用主动语态编写的吗？
- ☐ 需求有歧义吗？
- ☐ 使用了不常见术语吗？如首字母缩写、简称等。
- ☐ 输入和输出正确、详细吗？
- ☐ 需求真正表达了顾客想要的吗？

(5) 发送经修订改进的需求文档。更正需求错误之后，将需求文档发送给评审参与者，供他们进行单人评审或分组评审。

4.5.2 步骤 2：编写测试计划

软件测试计划是一份描述测试工作的目标、范围、方法和关注点的文档。编写测试计划是全面考虑需要完成哪些工作才能确认软件产品的可接受性的一种有用途径。写好的计划文档将帮助整个团队理解产品确认的原因和方法。测试计划应该编写得足够细致详尽，但不能让测试团队以外的人都看不懂。

编写测试计划需要完成以下任务。

- ☐ 为软件发布的质量目标排列优先级。
- ☐ 定义达到这些质量目标需要进行的测试活动。
- ☐ 评估测试活动对这些质量目标的支持情况。
- ☐ 指定为实施这些活动需要完成的操作。

(参见附录E，查看有关测试计划的例子。)

4.5.3 步骤 3：确定测试套件

完成测试计划之后，需求就是“可测试的”了。有一种将需求转化为测试用例的有效方案，那就是首先设计测试套件。测试套件也就是确认套件，是测试用例的集合，专门用作软件程序的

输入，表明程序有专门的行为集合。测试套件用来将相似的测试用例组织在一起，如订单处理。

测试套件通常包含每个测试用例集合的说明和目的，以及在测试中要用到的系统配置信息，可能还包含必备前提条件或步骤，以及对后续测试的描述。

测试套件文档（根据功能性）是一个有关测试用例内容的条理分明的表格，其中列出了所有测试用例的名称。套件可以通过列出主要的产品功能，然后列出每个功能相关的测试用例来组织，如表4-1所示（还可参见E.5节）。

表 4-1 功能与测试用例

[illegible]

组织套件的另一种方式是构建一个表格，表中的行表示业务对象的类型，列表示操作的类型（参见表4-2），每个单元格列出的是测试给定对象给定操作的测试用例。例如，“订单”就是“订单系统”的一个对象，对于下面CRUD（Create、Retrieve、Update、Delete）类型的每个操作，例如添加订单、罗列所有订单、编辑订单、删除订单、搜索订单等，“订单”这个业务对象都有相应的测试用例。“订单”的下一行可以包含“客户”业务对象和几乎所有这些操作的测试用例。

表4-2 测试套件标识矩阵

业务对象	操作类型				
	添 加	编 辑	搜 索	罗 列	删 除
订单	1. 创建一个网上订单 2. 创建一个POS订单 3. 创建一个目录订单 4. 创建一个经常性订单	1. 编辑一个网上订单 2. 编辑一个POS订单 3. 编辑一个目录订单	1. 通过客户ID搜索订单 2. 通过客户名字和地址搜索订单 3. 通过邮编搜索订单	1. 根据日期列出所有订单 2. 根据客户名字和地址列出所有订单 3. 根据州名列出订单 4. 根据产品列出订单	1. 删除一个网上订单 2. 删除一个POS订单 3. 删除一个目录订单 4. 删除一个经常性订单
客户	1. 创建一个零售客户	1. 编辑一个零售客户 2. 编辑一个批发客户	1. 通过客户ID搜索客户 2. 通过客户名字和地址搜索客户 3. 通过邮编搜索客户	1. 根据日期范围列出所有客户 2. 根据姓氏列出所有客户 3. 根据产品ID列出客户 4. 根据性别列出客户	1. 删除一个零售客户 2. 删除一个批发客户
账户
优惠券

使用结构清晰的列表或表格的好处在于能够获得全局认识,并且有助于找出那些需要花心思的地方。有一些业务对象和业务操作的测试很容易被忽略,例如“创建优惠券”。很显然,客户会使用优惠券,但测试人员容易忘记测试创建优惠券的能力,这样的话,测试套件文档中就会出现明显可见的空白。列表和表格可以清楚地提醒人们去注意被遗漏的测试用例,这样可以更快地改进测试套件,更准确地估计测试所需时间,找到更多的缺陷。这些好处能使被发现的缺陷尽快得到修复,使管理期望与现实情况一致。

4.5.4 步骤4: 命名测试用例

如果具备组织良好的系统测试套件,那么可以更便捷地列出测试用例,因为任务被分解成了许多个特定的子任务。

可能的确存在一些空白的列表项或单元格。如果套件的某部分从逻辑上看应该是有测试用例的,但你实在想不出来任何测试用例,那么你可以给它标上“TBD”(待定)作为说明。

测试用例的名字应该是一个描述测试情况的简洁短语。在需要采用不同的步骤来测试每种情况时,应该使用不同的测试用例。当步骤相同但需要的输入值不同时,可以使用同一个测试用例。

在构思测试套件的轮廓时,要考虑到应该出现但还未出现在软件需求规约里的功能或用例,在填写过程中还要注意在需求文档中可能遗漏的任何需求。

至此，你已经很好地了解了测试工作的范畴，差不多可以为测试用例排列优先级了。现在，你开始批判地审视自己的需求，可能你已经发现了那些被遗漏或不明确的需求，并且已经估计出你将能达到的基于规约的测试覆盖率。

4.5.5 步骤 5：编写测试用例描述及目标

在步骤4中，你大概已经生成了12个测试用例，这个数字还会随着测试工作的深入逐渐增长，从而让你的测试更加系统化。拥有大量测试用例通常可以提高覆盖率。

创建规模太大的测试套件不利的一面恰恰是它太大，因为需要花大量时间才能全面描述你计划好的每一个测试用例，而且结果文档也会非常庞大，难以维护。

对于每个测试用例，用一两句话来描述它的目标和目的。这种描述应该提供足够的信息，这样，当你在几周之后想起现在构思的某个测试用例时，就可以回头来查看详细信息。如果你真的面面俱到地编写了测试用例的详细步骤，那么任何一个团队成员都会按你的原意来实现测试。

要想做好描述工作，你就必须更细致地考虑每个测试用例。这样，在描述一个测试用例时，你可能会发现应该把它分解为两个测试用例，或者应该把它与另外某个测试用例合并。和其他步骤一样，你一定要注意是否存在未发现的需求问题。

4.5.6 步骤 6：创建测试用例

下一步是编写测试用例步骤，指定测试数据。在这一步中，测试技术可以帮助你定义测试数据和测试条件。每天创建大约10个测试用例是一个不错的准则。

着重考虑要求最为具体的测试用例，例如，选择覆盖以下要素的系统测试用例。

- 高优先级的测试用例或特性。
- 当前在测试中可用的软件组件。
- 在其他功能可以运用之前能够正常工作的特性。
- 产品演示或截图需要用到的特性。
- 需要明确的需求。

每个测试用例都应该尽量简单，成败分明。理想情况下，每个测试用例都有着统一的步骤：设置测试情形，通过特定的测试输入运行系统，验证系统输出的正确性。

高度可测试的系统通常有大量遵循“设置——运行——验证”模式的简单的测试用例。对于那些测试用例，可以用一种简单的单列格式来清晰地表达需要执行的步骤。但并非所有测试用例都简单，有时候，一次只测试一个需求是不现实的。有的系统测试可能有很长的场景，需要在每个步骤中运行好几个需求和验证正确性。对于这些测试用例，则需要使用双列格式。

在单列格式中，每个步骤就是一个简短的动词短语，描述的是测试人员应该执行的操作，例如，“输入用户名”、“输入密码”、“选择登入”、“见主页”。对期望输出的验证是通过动词“观察”和“验证”来编写的。如果需要多个输入，就必须验证多个输出。

在双列格式中，每个测试用例步骤有两个部分：测试输入是一个动词短语，描述测试人员在此步骤中应该完成的任务；期望输出是一个名词短语，描述测试人员在此步骤中应该观察的所有输出（参见附录E）。

如果对于某给定测试用例只有一个测试输入值，那么可以直接将这个数据值写进用到它的步骤中。然而，许多测试用例都有一组测试数据值，它们必须能够充分覆盖几乎所有可能的输入。这时需要定义和使用测试输入变量，每个变量在定义时都有一个由其选定值组成的集合，这些变量会用在测试用例的步骤中，就和在编程语言中使用的变量一样。在执行这些测试时，测试人员应该采用测试变量值每种可能的组合来多次重复执行每个测试用例，越多越好。

仔细挑选测试数据和定义测试用例步骤一样重要。边界条件和等价划分的概念是正确选择测试数据的关键。请试用下面的步骤来选择测试数据。

- 确定可能作为给定输入参数的所有输入值的集合。
- 定义有效输入值和无效输入值的边界。例如，负的年龄值是不存在的。你也可以检查明显不合理的输入。例如，年龄值为200肯定是不现实的。

（想要了解如何编写测试用例参见附录G，其中包含了39种测试技术。）

4.5.7 步骤7：评审测试用例

系统测试用例套件有利于发现许多缺陷，但也会遗漏不少未检测到的其他关键缺陷，一个有效的应对措施就是提高测试套件的覆盖率。

单元测试可以就其实现覆盖率进行评估，但系统测试用例则应该就其规约覆盖率来评估。实现覆盖率测量的是单元测试用例执行的代码行的百分比。如果有一行代码从没被执行过，那这一行就有可能存在未检测到的缺陷。

规约覆盖率测量的是系统测试套件中覆盖的已编写需求的百分比。如果有一个需求没在任何系统测试用例中被测试到，那么就无法保证能够满足这个需求。

你可以从两方面评估系统测试覆盖率：(1)测试套件本身是一个有关测试用例的结构分明的表格，因而很容易注意到系统中没被测试到的部分；(2)在各个测试用例中，可能输入值的集合应该包含所有输入值。

4.6 将用例转换为测试用例

Ivar Jacobsen创建的用例（use case）是一个描述参与者使用系统完成工作的场景。

下面是测试人员根据用例创建有效的测试用例（test case）的步骤。

4.6.1 步骤1：绘制用例图

测试用例可以用图4-4中的用例图来表示。

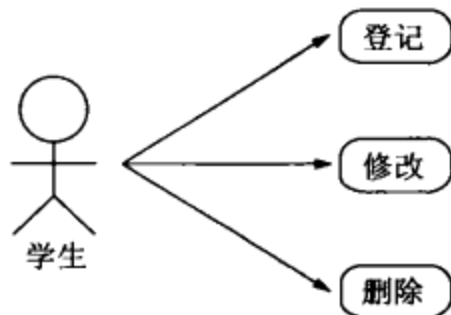


图4-4 用例图

椭圆代表用例，简笔人物画代表参与者，可以是人也可以是其他系统，线条代表参与者与用例之间的联系。用例是重要部分，每个用例代表将要实现的功能，每个参与者代表与功能交互的某个人或系统之外的事物。

4.6.2 步骤 2：编写详细的用例文本

以文本格式记录每个用例明细。表4-3所示是“登记”用例明细，由正常流和替代流组成。

表4-3 “登记”用例文本描述的格式

用例 ID	Enroll_001		
用例名称	登记一名学生		
创建者	John Doe	上次的更新者：	
创建的日期	3/15/2008	上次更新的日期：	
参与者	学生		
描述	将一名学生登记到班上		
触发器	学生希望在截止日期前登记		
前置条件	该大学已接收学生，登记时间已开始		
后置条件	学生的信息已通过验证并保存在学校登记系统里		
基本流	登记： 学生输入自己的名字 学生输入自己的地址 学生输入自己的电话号码 学生输入自己的学号 学生按下“提交”按钮 登记系统在下拉列表中列出可选课程 学生从下拉列表中选择一门课程并按下“接受”按钮 系统保存课程信息，并询问学生是否要选择其他课程 学生如果选择“是”，则登记系统继续执行（第6步），直到选完所有课程，并且学生按下“否”按钮 打印所有选定课程和时间表 学生登出系统		
替代流	A1：按下“提交”按钮（第5步），如果有任何信息不正确，则会在出错字段旁边显示出错消息，然后重复第5步 A2：学生按下“拒绝”按钮（第7步）		
包含	—		
优先级	高		
使用频率	根据需要而定		
业务规则	—		
特殊需求	—		
假设条件	所选课程的学生名额还未满		
注意事项	—		

4.6.3 步骤3：确定用例场景

一个用例场景就是用例的一个实例，是完成用例的一种完整“路径”。在执行用例中指定的功能时，系统的终端用户有许多种路线可循。图4-5所示是登记过程的一个流程图，直线表示的是基本（正常）路径。

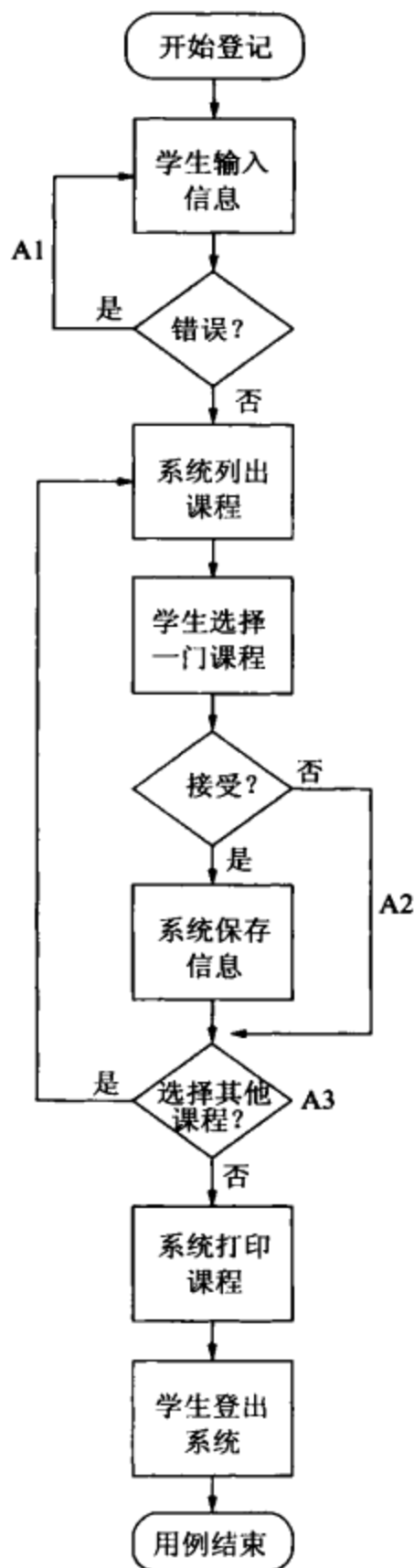


图4-5 登记流程图

A1和A2描述的是替代路径（或异常）。如果学生往系统中输入个人信息时出错，则为A1路径。如果学生已经选择了某个课程，然后又选择不接受它，这时就是A2路径。

表4-4列出了图4-5中可能的一些场景的组合。开始是基本流组合，然后添加替代流来定义各种场景，这些场景将被用作创建测试用例的基础。

表4-4 用例场景

场景 1	基本流			
场景 2	基本流	替代流 1		
场景 3	基本流	替代流 2		
场景 4	基本流	替代流 2	替代流 3	

4.6.4 步骤 4：生成测试用例

一个测试用例就是一套测试输入、执行条件以及针对特定目标而定的期望结果。

只要确定了一套场景，下一步就可以确定测试用例了，这一步是通过分析场景和评审用例文本描述来完成的。每个场景至少应有一个测试用例。对于每个无效的测试用例，应该只有一个无效的输入。

为了记录测试用例，可以使用矩阵样式，如表4-5所示。表中第一列是测试用例ID，第二列简要描述测试用例和待测试的场景，最后一列描述的是测试用例的期望输出，此外所有其他列包含的是用于实现测试的数据元素。V表示有效测试输入，I表示无效测试输入。

表4-5 登记测试用例矩阵

测试用例 ID	场景/条件	学生姓名	地址	电话号码	学号	被拒绝的课程	退出登记	期望结果
Enroll 1	场景 1：成功登记	V	V	V	V	否	是	显示选定的课程，退出系统
Enroll 2	场景 2：未经识别的学生	I	—	—	—	—	否	出错消息。退回到可选课程列表
Enroll 3	场景 3：拒绝某门课程	V	V	V	V	是	否	所选的课程被拒绝，退回到可选课程列表

4.6.5 步骤 5：生成测试数据

确定好所有测试用例后，对它们进行评审和确认，从而确保精确度，并识别冗余和遗漏的测试用例。如果测试用例得到批准，最后一步就是用实际数据替换表4-5中的I和V，如表4-6所示。有许多技术都可以用来指定数据值。

等价类划分和边界值分析是两种很不错的技术。（详情参见附录G。）

表4-6 登记测试用例细节

测试用例ID	场景/条件	学生姓名	地 址	电话号码	学号	选择的课程	期望结果
Enroll 1	场景1: 成功登记	John Doe	得克萨斯州达拉斯市布鲁克街2719号, 邮编为75093	(972)9832876	G3982	海洋学	显示课程和 timetable, 退出系统
Enroll 2	场景2: 未经识别的学生	无效	得克萨斯州达拉斯市布鲁克街2719号, 邮编为75093	(972)9832876	G3982	海洋学	报告出错消息, 退回到登录屏幕
Enroll 3	场景2: 未经识别的学生	John Doe	无效	(972)9832876	G3982	海洋学	报告出错消息, 退回到登录屏幕
Enroll 4	场景2: 未经识别的学生	John Doe	得克萨斯州达拉斯市布鲁克街2719号, 邮编为75093	无效	G3982	海洋学	报告出错消息, 退回到登录屏幕
Enroll 5	场景2: 未经识别的学生	John Doe	得克萨斯州达拉斯市布鲁克街2719号, 邮编为75093	(972)9832876	无效	海洋学	报告出错消息, 退回到登录屏幕
Enroll 6	场景2: 未经识别的学生	John Doe	得克萨斯州达拉斯市布鲁克街2719号, 邮编为75093	(972)9832876	G3982	无效	报告出错消息, 退回到登录屏幕
Enroll 7	场景3: 未经识别的学生	John Doe	得克萨斯州达拉斯市布鲁克街2719号, 邮编为75093	(972)9832876	G3982	拒绝海洋学课程	退回到登录屏幕

4.6.6 小结

用例在软件开发生命周期的前端很有用, 测试用例通常与生命周期的后期相关联。通过用例来生成测试用例, 测试团队可以在生命周期中尽早开始测试工作。

4.7 需求不存在或编写粗劣时怎么办

下面几节将概述没有良好需求时怎样创建测试用例。

根据具体的项目和组织, 需求可能编写得非常优良, 满足前面讲述的需求质量因素, 但也可能不够明晰, 有歧义, 这种情况就需要考虑采取其他替代方案。

4.7.1 即兴测试

1. 即兴测试的艺术

即兴测试是最不正式的一种测试技术, 因为其组织不够精密, 所以饱受非议。这种测试技术最常用作其他测试的候补方案, 在整个测试周期都可能用得上。在项目早期, 即兴测试可帮助测试人员理解程序, 研究项目。在项目中期, 获取的数据可以用于设置优先级和安排进度。当项目接近发布日期时, 即兴测试有利于更严格地检测缺陷修复, 这在前面介绍过。

即兴测试有它的长处——可以快速地找到重要的信息。它是即时执行的, 测试人员通过它采用任何可能合适的手段来寻找缺陷。它和回归测试不同, 后者是采用可重复的详细步骤来查找特

定问题，会得到明确的期望结果。

即兴测试在很多方面就像爵士乐即兴演奏一样，爵士音乐家有时候会使用一些由曲子领篇构成的改编歌曲集来即兴演奏曲子。在演奏完现有的可辨识的旋律之后，音乐家转而开始即兴独奏。有时候他们也会改变节奏，比如说，拖慢节奏。根据原版曲调，这种即兴独奏可能是可辨识的，也可能是不可辨识的。不过，在曲子的结尾，即兴演奏者通常会回到原来的旋律上。

软件测试与此类似，测试人员首先从一个记录好的测试设计入手，这个设计系统地描述了所有要覆盖的情况。进行即兴测试的一个有效方法是在一个房间里集合一群有能力的测试人员，让他们协同合作，进行即时测试。参与即时测试的测试人员查找缺陷的能力，与爵士乐演奏中的协作能力有很大的相似之处。

即时测试可以通过在已记录的现有测试基础上发展出其他一些测试方案来完成。

2. 即兴测试的优缺点

即兴测试的最大好处在于发现问题。很多时候，阅读需求或规约也无法了解程序行为，但即兴测试可能会找到测试策略中的漏洞，并且暴露出子系统之间不够明朗的关系，因此它可以用作检查测试完备性的工具。

在正式测试用例中，可能存在一些未发现的遗漏用例，但在随机测试中，它们将被找出来，就好像它们经过实际设置一样。在即兴测试中发现的缺陷通常是在某类被遗忘的测试用例中都会出现的缺陷。

即兴测试的另一个好处是为其他测试活动确定优先级。低级别功能和基本功能一般不会把即兴测试放到需求里，所以它没有相关的测试用例。

即兴测试的缺点在于这种测试形式不会记入文档，因此就不是可重复的，不会纳入到回归测试套件中。

4.7.2 探索性测试

1. 探索性测试的艺术

在需求和规约不完整或者时间不够的情况下，适合采用探索性测试。这种方式也可以用来验证前面的测试工作是否已经找到了最重要的缺陷。探索性测试和脚本测试结合执行是很常见的情况，例如，根据风险提前编写好一套用来测试软件的测试步骤。

作为测试计算机软件的一种技术，探索性测试不要求有非常严格的计划，它对文档比较宽容。探索性测试需要测试人员具备足够的引导测试的能力和知识，它使用反馈循环来引导和调整工作。James Bach说过：“对设计进行测试（也就是脚本测试）的经典方法就像是提前写好所有问题的‘问答游戏’。”

探索性测试是一种测试战术，它首先假设软件存在某些错误和缺陷，然后对之进行检测。在这种测试方法中，学习、测试设计和测试执行同步进行。在测试软件时，测试人员会学习一些知识，然后凭借这些知识和自己的经验与创造力生成新的测试。

探索性测试已被人们采纳了一段时间，它和即兴测试也有相似之处。在20世纪90年代早期，即兴测试通常是一项草率的工作。这个术语最早是由Cem Kaner在他的著作*Testing Computer Software*中提出的。探索性测试比经典即兴测试的结构更加严谨，和任何其他智力活动一样严格。

探索性测试的目的是查明软件的工作情况，询问应如何处理困难的和简单的情况。这种测试依赖于测试人员开发测试用例和找出缺陷的能力。测试人员对产品和各种测试方法了解得越多，测试就会进行得越好。

执行探索性测试时，没有准确的期望结果。测试人员负责决定要验证哪些，并严格检验结果的正确性。

事实上，探索性测试和脚本测试几乎总是结合进行的，至于更倾向于哪个，则取决于上下文。

根据Cem Kaner和James Bach的思想，与其说探索性测试是一种方法论，不如说它是一种理念体系。探索性测试的文档记录的内容包括已执行的所有测试和bug。

2. 探索性测试的过程

下面是探索性测试的基本步骤。

- (1) 确定产品的目标。
- (2) 确定功能。
- (3) 确定可能存在不稳定性的区域。
- (4) 测试每个功能，记录问题。
- (5) 设计和记录一个统一的验证测试。

James Bach说过：“根据我的实践，探索性测试的执行遵循着特别的计划，但并不是很严密的计划……没有编写详细的脚本。如果下一个测试受到上一测试结果的影响，那么我们执行的就是探索性测试。当我们无法说出哪种测试应该在测试周期中提前执行时，我们的探索就更深一层了。”

测试用例本身是无法提前计划的。

- 探索性测试可以与产品开发和测试执行同时进行。
- 这种测试的基础是隐式规约和显式规约，以及产品构造要求。
- 探索性测试首先假设某个推测是正确的，然后探测这个推测是否可行的证据。
- 以某种心智模型为基础。
- “试试看是否行得通。”

3. 探索性测试的优缺点

探索性测试的主要优点是可以少做准备，以最快的速度找到最重要的bug，仿真比脚本测试更理智。

另一个优点是测试人员可以根据前一测试的结果进行演绎推论，实时指导将来的测试。他们在开发目标更多的环境之前，不用进行一系列的脚本测试。理智地执行探索性测试可以更快检测到bug。

此外还有一个优点：在初步测试之后，大部分bug就已通过某种探索性测试被检测出来。这可以如是总结：“通过了某些测试的程序趋向于通过其他相同的测试，而且很有可能在还未开发的其他测试或场景中失败。”

探索性测试的缺陷在于，测试不能提前评审（因此不能在代码和测试用例中防止错误发生），而且很难准确展示已经执行了哪些测试。

当重复执行探索性测试时，它们不是按照完全一样的方式进行的，如果想知道究竟是哪种功能性测试，那么这就是一个很大的缺点。

5.1 Edward Deming 的贡献

尽管Henry Ford和Fredrick Winslow Taylor在制造业生产方面做出了巨大的贡献，但Edward Deming博士已经超过了他们。Deming影响着各行各业的每个方面，包括政府、学校和医院等。他对于人们怎样思考，怎样看待自己，怎样与客户、他人以及社会进行相处等都有深远的影响。

1928年Deming获得了物理学博士学位，并在随后的4年里发表了有关电子对材料结构的影响的论文。他在物理学领域开创了他的事业。1934年他开始从物理学和物理化学等研究领域转移，并发表了他在统计学领域的第一篇论文。1937年他写了一篇关于误差的统计理论论文。

根据美国法律要求，美国联邦政府每十年要进行一次人口普查，1940年Deming参加了商业部人口普查署的活动。统计学是完成这项任务最合适的工具，所以我们发现他的26篇系列论文几乎都在解决抽样方面的问题。他于1944年发表的一篇论文中向工程师介绍了Shewhart的质量控制方法。他带头把这一方法引入到工程师的战时培训，并在斯坦福大学亲自讲了第一堂课。大约从1945年以后，人们就不再把Deming看做一个物理学家，而把他当成统计学家了。所以，在1948年当麦克阿瑟将军需要在日本进行人口普查时去拜访Deming就不足为奇了。1953年——在他开始与日本管理人员共事的3年后——他开始投身于质量管理理论的研究，并把研究成果介绍给了美国的管理人员。1953年他发表了*Management's Responsibility for the Use of Statistical Techniques in Industry*，从而开创了他在随后40多年研究的主题。他在日本已经看到了这种变革。

75

5.2 统计方法扮演的角色

Deming的质量方法包括了统计方法的使用，因为Deming相信，当过程中出现了偏差时，统计方法对于把混乱减到最小是非常必要的。统计学还能帮助我们了解过程本身、增进控制并改进过程。有句话说得很透彻：“我们只相信数据。”当问题解决以后，对于引起质量显著提高的原因的定位总会受到特别关注。Deming指出许多统计技术并不难，但需要有一点点的数学背景。然而，教育是一个非常有力的工具，在一个组织的各个阶层都需要，以保证其正常运转。

下面是对一些统计方法的概述，这些方法在后面的章节中还会做进一步的描述，并会应用到软件测试中。更详细的描述请参见第三部分。

5.2.1 因果图

因果图也常常被称为“鱼骨图”，在头脑风暴法中可以用来找出影响局势的各种因素。因果图是通过表示某些结果与可能的起因之间的关系来确定问题发生的可能起因的工具。

5.2.2 流程图

流程图是将过程文档化的一种图形化方法。它是一张显示过程或工作流的顺序执行步骤的图，执行这些步骤将开发出产品或服务。使用流程图的理由是：要改进过程，就需要首先理解流程图。

5.2.3 帕累托图

帕累托图是一种普遍采用的图形化技术，它会把要分析的事件都加以命名。在帕累托图中，事件是按名称来计数的，在柱状统计图表中事件按照出现的频率以升序排列。帕累托图的分析采用80/20规则。例如，当公司20%的客户带来了80%的收入时，就应该重点关注这20%的客户。

5.2.4 运行图

运行图是一种图形化的技术，其图形数据是以年代顺序标明的，以展示当前测量的某个特征的变化趋势，从而找出潜在的原因而不是随机的变化。

5.2.5 直方图

直方图是对测量的数值的一种图形化描述，按照数据出现的频率或相对频率来组织数据。此外，直方图还能提供数据的平均值和偏差。

5.2.6 散布图

散布图是用来显示两个变量之间或两个变化因素之间关系的图形。

5.2.7 控制图

控制图是一种统计方法，用来区分由过程展示出来的特殊变化和常规变化。它实际上是一个运行图，只是在过程平均值的两侧分别画有统计得出的上限和下限。

5.3 Deming 的 14 条质量原则

Deming概括了14条质量原则，为了达到质量要求，必须同时使用这些原则。尽管这些原则过去是用于工业界的，对政府、学校和医院都产生了很大的影响，但从信息技术的观点来看，也有很多可以用于提高软件质量。下面会对每一条质量原则做简要的讨论，然后描述质量保证组织如何运用这些原则。

5.3.1 第 1 条：树立始终如一的目标

许多公司往往只是停留在眼前的问题上，而对未来缺少足够的重视。按照Deming的说法：“热衷于解决当前的棘手问题，从而在将来变得效率越来越高是一件很容易的事，但如果没有长远的计划，没有哪一家公司能够在市场上立得住脚。”始终如一的目标需要创新（例如长期的规划等），需要在研究和教育上投资，同时还需要持续改进产品和服务。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 制定一个质量保证计划，以提出长远的质量方向。
- 需要软件测试人员为每个项目开发和维护全面的测试计划。
- 鼓励质量分析师和测试人员提出创新的想法，以最大限度地提高质量。
- 努力做到质量过程的持续改进。

5.3.2 第 2 条：采用新的质量观念

质量必须成为一种新的信仰。Deming说：“举例来讲，生活成本与一定数量的金钱能够买得到的商品和服务成反比。可靠的服务将降低成本，延期和错误将增加成本。”消费者将不再为商品和服务的延期和错误埋单，因为那样会降低他们的生活水准。系统中可接受的缺陷水平的容许范围是位于质量和产量之间的障碍，即匆匆验证质量可能会降低质量级别。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 对信息技术组织进行有关质量的需求和价值的教育。
- 把质量保证部门的重要性提升到与其他部门一样的水准。
- 摒弃那种认为质量保证是消极的“监控器”的观点。
- 制定风险管理计划，并拒绝接受任何超出可接受风险容许范围的异常情况。

5.3.3 第 3 条：停止对大量审查的依赖

把差的质量检查出来是一种陈旧的想法。当很难确定缺陷会在过程中的什么地方发生时，比较好的做法是检查我们现在正在怎样做，而不是只检查最后的产品。提高质量不应该依赖大量的检查。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 为了在整个开发周期中达到质量标准，要引入并强化使用技术评审、走查和审查等非防御性的技术。
- 给整个组织慢慢地灌输质量意识，并把这项活动当成一个切实的、可度量的工作产品交付物。
- 需要信息技术质量的统计学依据。

5.3.4 第 4 条：结束仅靠价签来激励企业的实践活动

“如果同一个项目有多家供应商，则会使恶行增加，这种恶行是与生俱来的，对任何一个供应商来说都是非常糟糕的。”采购员应该与某一个供应商建立长期的忠诚与信任关系，从而给公

司提供最好的服务。不应只用标准手册来约束供应商必须达到商业要求，而还应该在供应商管理中积极地引入Deming的14条质量原则。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 需要软件质量和测试供应商提供质量的统计依据。
- 为每一个质量保证工具、测试工具或服务选择最好的供应商，并建立与质量计划一致的工作关系。

5.3.5 第5条：坚持不懈地、永久地改进生产和服务系统

改进不是一时的努力，管理层有责任持续不断地改进质量。“救火不是改进。找出失控点，找到特定的原因并予以排除只是把过程恢复到原来的位置。改进的责任是一个永不停止的过程。”

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 持续不断地改进质量保证和测试过程。
- 不能依赖主观判断。
- 应用统计技术，例如因果分析方法，来找到问题的根源，并进行测试分析。

5.3.6 第6条：组织培训及再培训

因为所受培训很少，甚至没有培训，员工根本不知道什么时候已经正确地完成了工作。很难排除不恰当的培训。Deming强调，只要绩效的统计情况不在控制范围之内，并且还需要学习新内容，培训就不应该停止。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 组织进行现代培训活动及实践。
- 鼓励质量人员通过参加研讨会和上课等方式不断地积累有关质量及测试技术的知识。
- 对举办新的研讨会及成立特殊兴趣小组的员工予以奖励。
- 应用统计技术确定何时需要培训及培训完成时间。

5.3.7 第7条：确立领导职责

“把人们安排到他们根本不适应的岗位上是不可原谅的。大部分所谓的‘吊儿郎当’（有些人看上去好像很懒，工作起来好像不认真）的人通常是被安排在错误的工作岗位，要不然就是管理非常糟糕。”发现抑制因素是管理人员的责任，这些抑制因素会妨碍员工为自己的工作感到骄傲。从信息技术的观点来看，开发人员经常把质量工作看成是质量保证部门的责任。质量保证部门作为质量主管机构应该积极进取，并应指出质量是每个人的责任。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 如果质量保证人员测出了过多的缺陷，就要花时间对相应的开发人员进行有关如何有效地对代码进行单元测试的培训。
- 加大监管力度，这是管理部门的责任。
- 允许项目负责人花更多的时间协助员工工作。
- 用统计方法指明哪里有错误。

5.3.8 第 8 条：驱除恐惧

人们通常没有解决问题的动机，并且会觉得提出新想法太危险，因为人们总是害怕失去加薪、升职或工作的机会。“恐惧带来了可怕的代价。恐惧无处不在，它剥夺了人们的自尊，伤害了人们，剥夺了人们为公司做贡献的机会。当人们不再恐惧时，所发生的一切将是令人难以置信的。”对审查的恐惧是一个普遍问题。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 倡导“质量是美德，应该受到奖励”的思想，鼓励提出各种新的想法来提高质量。
- 在结构化走查、审查或JAD会话之前，应该确保每个人都能理解基本规则，并营造一种“无私”的环境。
- 定期安排“质量日”，让大家无拘束地分享质量改进的想法。

80

5

5.3.9 第 9 条：打破员工区域之间的壁垒

当各部门的目的不同，并且不能作为一个团队来共同解决问题、制定政策或确定新的发展方向时，就会出现大量的问题。“人们在各自的部门工作都很杰出，但如果不同部门的目的相抵触，就可能毁了公司。所以要有团队合作的精神，大家共同为公司工作。”

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 促进质量保证部门和其他部门（尤其是开发部门）紧密合作，质量保证部门应该被看成“好人”，因为其目标是尽力使软件产品质量出类拔萃。
- 指出应努力在产品交付生产之前发现缺陷，使之不会再被用户发现。

5.3.10 第 10 条：解除工作口号、训词及目标

“口号从来不会对任何人做好工作有所帮助，而只会带来挫折和抱怨。”诸如“零缺陷”或“第一次就做对”之类的口号表面看上去很好，但问题在于这些口号往往被看成了管理层不理解员工、不关心员工的问题和烦恼的信号。往往是在设定目标，而不去描述如何完成这些目标。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 鼓励管理层避免使用口号。
- 制定质量标准、规程和过程并文档化，供公司的其他部门使用，以利于最大限度地提高质量，而不需要创造很多口号。

5.3.11 第 11 条：去除数字化目标

“配额或其他工作标准，如可度量的日工作量或效率等，可能会比其他任何一个工作条件都更多地阻碍质量的提高。因而这些工作标准的普遍使用，必然会带来低效率和高成本。”正确的工作标准应该定义哪些是质量要求可接受的，哪些是不可接受的。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 不仅要看数字，还要仔细地看质量标准。
- 避免正式地发布个人或部门的缺陷率。

81

- 为了提高质量，应该与开发部门合作定义质量标准和规程。
- 当出现具体的质量问题时，应该由部门经理非正式地进行宣布。

5.3.12 第12条：消除阻碍员工自豪感的壁垒

把人看成是一种商品，需要的时候就重用，不需要的时候就辞退，让他们重新回到劳动力市场上。经理们虽然解决了很多问题，但总是倾向于回避人的问题。他们经常会形成“质量控制循环”，但这通常也是经理们假装正在解决某个问题的办法。管理层很少授予员工任何职权，也不对他们的建议采取任何行动。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下两点。

- 慢慢地灌输一种印象，即质量就是他们的交付物，而且是一种非常有价值的商品。
- 把职责委派给员工，让他们去定义什么是高质量并想办法努力完成。

5.3.13 第13条：开设有关教育和再培训方面的强有力的课程

人们必须不断获得新的知识和技能。教育和再培训是对人的投资，需要进行长远规划。教育和培训的目标是使人们适应新的工作和责任。

为了很好地应用这一原则，信息技术质量保证组织应该做到以下几点。

- 鼓励质量人员通过参加研讨会和上课等方式不断地积累有关质量及测试技术的知识。
- 对举办新的研讨会及成立特殊兴趣小组的员工予以奖励。
- 为员工掌握新的质量技能进行再培训。

5.3.14 第14条：采取行动完成转变

最高管理层需要推动上述13条原则的贯彻。包括经理在内的每一个员工都需要想清楚怎样才能不断地提高质量，但这件事必须由最高管理层来发起。下面讨论的这个过程就可以应用Deming的14条原则。同时它也是本书不断强调的用来改进软件测试过程的一个过程。

82

5.4 通过“计划、执行、检查、改进”实现持续改进

术语“控制”有很多不同的含义，包括监管、统治、调节或抑制等。“质量控制”中的“控制”是指定义工作目标，制定和执行计划以达到目标，检查并确定是否实现了预期的结果。如果没有实现预期的结果，就要修改工作规程以完成计划。

“Deming循环”（或PDCA循环，见图5-1）是描述上述过程的一种方法，在日本是以Deming命名的，因为是Deming把这种方法引入了日本，尽管此方法最初是由Shewhart提出来的。Deming循环，加上Deming的其他管理理论，构成了日本制造业起死回生的基础。“管理”一词描述了许多不同的功能，包括政策管理、人力资源管理、安全控制以及组件控制和材料、设备和日常进度的管理等。本书中，Deming模型主要应用于软件质量。

在循环的“计划”象限中，可以定义目标，并确定要达到这些目标所需要的条件和方法。在这个阶段清楚地描述目标和达到目标所需的政策是至关重要的。如果可能的话，应该用数字来记录特殊的目标。而且还应该描述达到目标所需的手段和方法的规程及条件。



图5-1 Deming质量循环

在循环的“执行”象限中，要创建条件，并为执行计划举办必要的培训。每个人都透彻理解目标和计划是极为重要的。要教会员工掌握规程和技能，这些是他们完成计划和透彻理解工作所必需的。然后工作才能够按照既定的规程进行。

在循环的“检查”象限中，人们必须通过检查来确定工作是否是按计划进展的，以及预期的结果是否已达到。必须针对条件的变化或可能出现的异常对系列规程的运行进行情况检查。应该尽可能经常将工作结果与目标作一比较。如果检查发现了异常（也就是说，如果实际值偏离了目标值），那么就要开始研究异常的起因，以预防异常的再次出现。有些时候，有必要对工人进行再培训，也有必要修改规程。确保反映出这些变化，并在下一次计划时更加充分地考虑，这是非常重要的。

在循环的“改进”象限中，如果检查出工作没有按计划进行或结果不是预期的，就必须拿出措施以采取适当的行动。

5.5 遵循 PDCA 循环

上述流程不仅能保证产品质量满足期望，而且能保证预期的价格和交付日期都是令人满意的。有时会由于要处理一些当务之急的事情使我们不能达到最佳的结果。遵循PDCA循环，我们就能够改进我们的工作方法，获得期望的结果。重复使用PDCA将使得改进工作质量、工作方法及工作结果成为可能。有时会用一个上升的螺旋来描述这一概念，如图5-2所示。

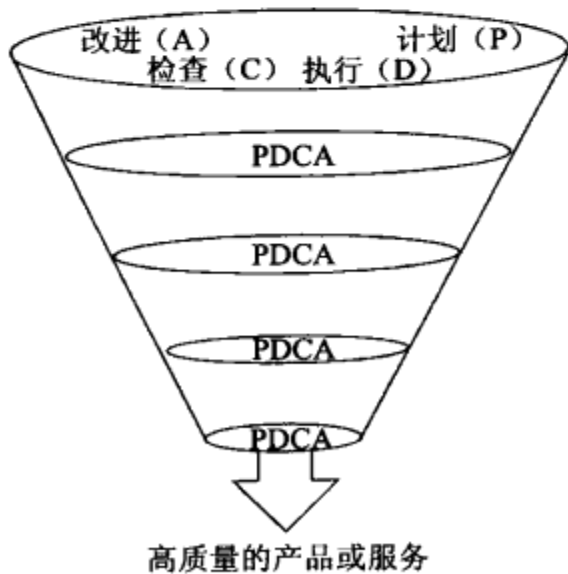


图5-2 上升的螺旋

瀑布测试概述

瀑布生命周期开发方法是由从需求到编码等多个不同的阶段组成的。生命周期测试是指测试与开发生命周期并行实施，并且是一个持续过程。通过运用质量循环、质量理论及统计技术就可以把Deming的持续改进过程应用到软件测试中。

从生命周期测试的心理学角度讲，测试最好在开发组织之外进行。这样做的动机是可以更加清晰地定义需求，并且测试组织作为第三方可以更有效地验证这些需求。

测试计划是软件测试的圣经，是一个用来指定测试目标、测试范围、策略方法及测试细节的文档。对于如何制定一份好的测试计划，有专门的方针可以提供指导。

在每个生命周期阶段的两大质量保证验证手段是技术评审和软件测试。技术评审更具有预防性，也就是说，它的目的是尽早地排除缺陷，而软件测试则是验证已经开发完成的实际代码。

本部分的目标如下：

- 讨论生命周期测试是如何与开发并行活动；
- 描述怎样应用Deming的过程改进原则；
- 讨论生命周期开发与测试的心理学；
- 讨论一个良好的测试都包括哪些内容；
- 列举和描述技术评审与测试是怎样的验证技术。

下面是对瀑布式生命周期开发方法及相关测试活动的一个概述。在技术评审和测试技术中应用了Deming的持续质量改进原则。

6.1 瀑布式开发方法

生命周期开发或瀑布式方法把开发周期划分成了几个离散的阶段，每一阶段都有严格的、顺序的开始和结束（见图6-1）。每一阶段都要在下一阶段开始之前充分地完成。从开发过程的理论上讲，一旦某个阶段已经完成，就再也不能回过头来修改。

从图6-1中可以看到，瀑布式开发方法的第一个阶段是用户需求。在这个阶段，要会见用户，分析用户的需求，要编制文档来详尽描述用户的需求。任何重构或过程的再设计都要合并到这一阶段。

在下一个阶段，要从数据和功能的角度出发，绘制实体关系图、过程分解图及数据流图，以便于把系统分解成可管理的组件。逻辑设计阶段的输出可用来开发系统的物理设计。在物理设计和程序单元设计阶段，会应用各种不同的结构化设计技术，如数据库模式、Yourdon结构图、Warnier-Orr图等，编写用于下一阶段的设计规约。

在程序单元设计阶段，程序员根据前一阶段产生的物理设计来开发系统。一旦完成，系统将进入编码阶段，这时就要用编程语言来写系统，就要进行单元或组件测试、集成测试、系统测试及最终用户测试，最终用户测试常常也称为验收测试。

现在把应用系统交付给用户，进入操作和维护阶段（图6-1中没有表示）。这时还会发现并更正各生命周期阶段中产生的缺陷，并把新的改进归并到应用系统中。

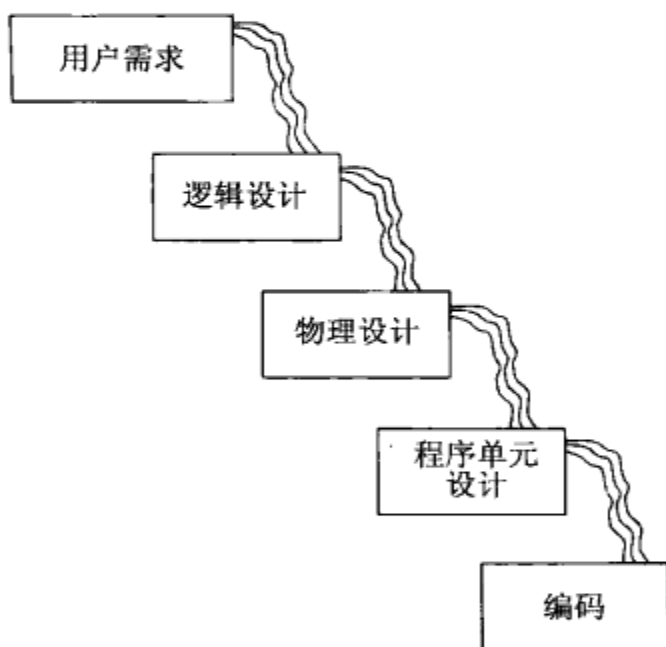


图6-1 瀑布式开发方法

6.2 “阶段化”持续改进方法

第5章中讨论的Deming持续改进过程可以有效地应用到瀑布式开发周期中，即采用Deming

的质量循环（即PDCA），也就是计划、执行、检查和改进。主要是从软件测试和质量控制或技术评审两个方面加以应用。

正如在第一部分中所定义的，质量保证的3大组成部分包括软件测试、质量控制和软件配置管理。软件测试的目的是通过验证和确认活动保证软件设计、代码和文档能够满足相应的需求。软件测试的主要环节包括测试计划、测试设计、测试开发及测试执行等。质量控制是用来监控工作和观察需求是否实现的过程和方法，它侧重于通过结构化走查和审查来排除软件开发生命周期中产生的缺陷。

88

6.3 生命周期测试的心理学

在瀑布式开发生命周期中，典型的一致做法是把测试部门和开发部门分开。通常，测试组织与开发组织会汇报给不同的领导。这样做的根据在于需求文档和设计文档都是在开发生命周期的特定阶段形成的，单独的质量保证组织应该能够把这些文档转换成测试计划、测试用例和测试规约。这里隐含的假设是：程序员不应该测试自己编写的程序，负责程序设计的组织也不应该测试自己开发的程序。

有人认为，软件测试是一个破坏性的过程，对一个程序员来讲，要从开发软件产品突然转变到试图去发现缺陷或破坏软件是很困难的。人们认为，程序员不可能有效地测试他们自己编写的程序，因为他们不会去暴露自己的错误。

该论点的另一个理由是，程序员对程序需求的错误理解可能会带来程序上的错误。这样，程序员在测试他们的代码时可能会存在同样的偏见，而不能像其他人那样有效地进行测试。

让程序员测试自己的程序是不可能的，因为他们通常有思维定势；如果让一个对程序没有思维定势的人来测试可能会更有效。既然开发部门的交付物都已经文档化了，为什么不让其他人来验证呢？

有人认为，衡量一个开发组织是否有能力主要是看他们能否按时、经济地开发出程序或系统。正如单个的程序员难以做到客观那样，让开发组织做到客观也是很困难的。如果要通过共同努力去发现了尽量多的缺陷，那么可以想象，这个项目很可能会延期或增加成本。其结果一定是质量不高。

从实践的观点看，软件产品的质量应该由一个独立的组织来负责，应该成立产品测试或质量保证这样的组织作为独立的部门来提供服务。

6.4 将软件测试作为持续改进过程

软件生命周期测试是指测试与开发周期并行实施，并且是一个连续的过程（见图6-2）。软件测试过程应该在应用程序生命周期的早期就开始，而不是在传统的编码阶段完成之后的确认测试阶段才开始。测试应该与应用程序的开发集成在一起。为了实现这一点，需要开发组织做出承诺，与质量保证组织密切沟通。

89

测试计划是在需求阶段开始制定的，它要对测试工作进行组织。测试计划文档主要描述在将要进行的测试活动中采用的方法，包括要测试的项目、要执行的测试类型、测试日程表、人力资源、汇报流程及评估标准等。

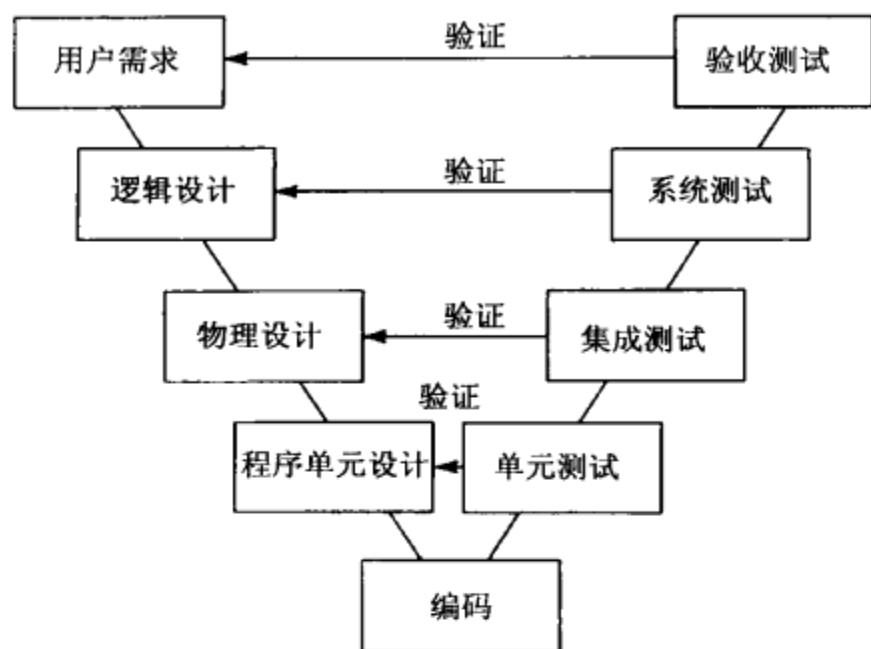


图6-2 开发阶段与测试类型

在逻辑设计、物理设计和程序单元设计过程中，要进一步地细化测试计划，并且要编写测试用例。测试用例是测试数据和测试脚本的一个特定集合。测试脚本在测试过程中引导测试人员，以保证测试在分开执行时具有一致性。测试还包括用来检验测试目标是否正确达到的预期测试结果。测试脚本和测试数据在编码阶段产生。在应用程序测试阶段，要运行测试脚本，并对结果进行分析。

图6-2显示了应用程序开发与测试活动的对应关系。应用程序开发周期是从用户需求和设计一直进行到代码完成。在进行测试的设计和开发时，要在测试计划中制定验收测试的标准。随着过程不断地细化，要制定系统测试、集成测试及单元测试的需求。对每种不同的测试类型，可以制定单独的测试计划，也可以共用一个计划。

在执行测试时，上述过程是颠倒过来的，即从单元测试开始。执行集成测试是把单个的单元测试过的代码段组合在一起进行测试。集成测试一旦完成，就会从系统整体出发，对系统进行测试，即所谓的系统测试。系统测试是一个多方面的测试，要评估系统的功能、性能及易用性。最后是验收测试，是用户运行的测试，用来检验系统是否可以满足用户的最初目标 and 需求。在某些情况下，就用系统测试作为验收测试。

回想一下，PDCA（即计划、执行、检查、改进）方法是用来控制、监督、管理、调节或抑制一个系统的控制机制。这一方法首先定义了过程的目标，制定并执行计划来实现这些目标，然后通过检查来确定是否达到了预期的效果。如果没有达到，就要修改计划来完成目标。可以将PDCA质量循环应用到软件测试中。

当持续改进过程的“计划”步骤应用于软件测试时，是从定义测试目标开始的，例如测试结果是什么？测试标准远不止简单地保证软件运行符合规约，其目的是要保证所有相关的人员都参与定义测试标准，以最大限度地提高质量。

这一步骤的主要交付物就是软件测试计划。测试计划是完成测试的基础。应该把测试计划看成一个持续变化的文档。当系统发生变化时，计划也要跟着改变。测试计划还应该成为应用程序

交付给用户之后的系统维护文档的一部分。一份好的测试计划的主要内容包括概述、整体计划及测试需求。随着越来越多的细节得以确定，还要增加业务功能、测试日志、问题及总结报告、测试软件、硬件、数据、人员需求、测试日程表、测试进入标准及退出标准等内容。

当持续改进过程的“执行”步骤应用于软件测试时，需要描述怎样设计和执行测试计划中包含的各项测试。测试设计的内容包括测试用例、测试规程及脚本、预期结果、功能/测试用例矩阵、测试日志等方面。测试计划编写得越明确，测试设计就越容易进行。如果在测试计划制定之后、测试执行之前，系统发生了变化，那么要相应地更新测试计划；也就是说，无论什么时候系统发生了变化，测试计划都要跟着改变。

测试团队负责测试的执行，并要保证测试按照计划执行。“执行”步骤的要素包括选择测试工具、定义资源需求、定义测试设置条件及环境、测试需求及应用程序的实际测试等。

当持续改进过程的“检查”步骤应用于软件测试时，主要包括评估测试过程的进展如何。需再次强调的是，统计学家的“我们只相信数据”信条对于Deming方法来说也是至关重要的。收集精确、及时的数据作为尽可能多的决策的依据是非常重要的，关键的数据是测试度量(metrics)，如缺陷的数量和类型、付出的工作量及进度表的状态等。

出具测试报告也是非常重要的。测试从设定目标开始，然后要确定功能，选择测试用例以验证测试功能，创建测试条件并执行测试。测试团队为了撰写测试报告，必须正式记录测试结果，并与测试计划和系统目标关联起来。从这种意义上讲，测试报告把前面所有的测试任务的执行顺序都颠倒过来了。

在测试结束时及关键的测试检查时间点上，要分别编写总结测试报告及中间测试报告。编写报告的过程对于中间测试报告和总结测试报告而言是一样的，正如其他的测试任务一样，编写测试报告也要进行质量控制，即应该进行评审。一份测试报告至少应该包括发现缺陷的记录、数据精简技术、根本原因分析、形成测试结论、就改进测试过程给管理部门提的建议等。

当持续改进过程的“改进”步骤应用于软件测试时，主要包括对那些没有按照计划执行的工作或者与计划预期的结果不一致的工作要想办法采取适当的行动加以改进。分析结果要反馈在测试计划中。例如，要包括更新测试套件、测试用例、测试脚本，以及重新评估测试的人员、过程和技术维度等。

6.5 测试的圣经：软件测试计划

测试计划是用来描述在将要进行的测试活动中需要采取的方法的文档，而且还会作为质量保证测试职能部门与其他相关部门（如开发部门）之间服务级别上的协议。应该在开发周期的早期就制定测试计划，以增进分析、设计和编码等活动之间的相互配合。测试计划要定义测试的目标、范围、策略和方法、测试规程、测试环境、测试完成标准、测试用例、要测试的项目、要进行的测试、测试进度安排、人力资源、汇报流程、项目假设、风险及应急计划等。

制定测试计划时应该确保简单、完整、最新，并为相关人员所接受，以得到反馈和认可。一份出色的测试计划应该逻辑清晰，把冗余测试减到最少，并且能够证明覆盖了全部功能，提供可用于监控、跟踪及汇报测试状态的流程，包含对所涉及部门的角色和职责的清晰定义，具有目标交付日期，并将测试结果清晰地文档化。

制定测试计划有两种方式。第一种方式是制定一个主计划，提供对每一个详细测试计划的概述，即测试计划的测试计划。每个详细测试计划则用来验证瀑布式开发生命周期中某个特定的阶段。例如，测试计划包括单元测试计划、集成测试计划、系统测试计划及验收测试计划等。其他详细测试计划包括应用程序升级、回归测试及程序包安装等。单元测试计划是面向代码的，非常详细，但由于范围有限而非常短。系统测试计划或验收测试计划主要关注整个系统的功能测试或黑盒测试，而不是某个软件单元。详见E.1节及E.2节。

92

第二种方式是只有一个测试计划。这种方式在一个测试计划中包含了所有的测试类型。虽然通常称为验收/系统测试计划，但实际上包括了单元测试、集成测试、系统测试、验收测试及为完成测试要考虑的所有事情。

通常在测试规程部分，测试计划的主要组成是测试用例，如图6-3所示（也可参考E.8节）。测试用例定义了执行测试时所依据的一步一步的过程。测试用例包括测试的目标和条件、启动测试所需的步骤、数据输入、预期结果及实际结果。同时还要提供诸如软件、环境、版本、用例ID、检查及测试类型等其他信息。

日期: _____	测试人: _____
系统: _____	环境: _____
目标: _____	测试ID: _____ 需求ID: _____
功能: _____	界面: _____
版本: _____	测试类型: _____
(单元测试、集成测试、系统测试、验收测试)	
要测试的条件:	

数据与要执行的步骤:	

预期结果:	

实际结果: 通过_____ 失败_____	

图6-3 测试用例格式

6.6 制定测试计划的主要步骤

测试计划是完成测试的基础，应该被看成一个“活”文档，也就是说，当应用程序发生变化时，测试计划也要跟着改变。

一份好的测试计划总是鼓励“质量走在设计和编码之前”这样的观点，它应该能够证明已经覆盖了全部的功能，并且测试用例能够追踪要测试的功能，它还包含监控和跟踪发现的缺陷及报告状态的可行机制等。E.2节是一个系统/验收测试计划的模板，把单元测试、集成测试和系统测试计划都组合在了一起。本节描述在瀑布式生命周期开发方法中怎样制定测试计划。

下面是制定一份出色的测试计划需要完成的主要步骤。

6.6.1 步骤 1：定义测试目标

规划任何测试的第一步都是要确立究竟要完成什么工作才算测试取得了成果。这一步要保证所有相关人员都参与到将要使用的测试标准的定义中来。测试计划的制定人员要确定测试将要完成的内容、将要执行的特定测试、测试期望值、测试的关键成功因素、将要执行的测试的约束条件和范围、测试期望的最终产品、最后的系统总结报告（见E.11节），及最终的签署和批准等内容。测试目标需要进行评审，并获得批准通过。

6.6.2 步骤 2：确定测试方法

测试计划的制定人员要概括出整体的测试方法或怎样执行每一个测试。这包括将要使用的测试技术、测试进入标准、测试退出标准、与开发部门协调测试活动的规程、测试管理方法，例如，缺陷报告和跟踪、测试进展跟踪、测试状态报告、测试资源和技术、风险及测试基础的定义（功能需求规约等）。

6.6.3 步骤 3：定义测试环境

测试计划的制定人员要检查物理的测试设施，定义硬件、软件和网络，确定需要的自动化测试工具和支持工具，定义需要的桌面帮助支持，准备测试需要的专用软件，并制定一个计划来支持上述各项内容。

6.6.4 步骤 4：制定测试规约

测试计划的制定人员要求测试团队撰写测试规约，制定测试规约的格式标准，分解工作任务，并把任务分配给团队成员，确定要测试的产品特性。测试团队要把每一个特性的测试规约文档化，并且与功能规约建立交叉索引。另外，还要确定测试规约之间的相互依赖关系及 workflow，并要对测试规约进行评审。

6.6.5 步骤 5：确定测试日程表

测试计划的制定者要基于资源的可用性和开发日程表来确定测试日程表，要比较日程表与最后期限，平衡资源与需要的工作量，定义主要的检查点，并制定应变计划。

6.6.6 步骤 6：评审及批准测试计划

测试计划的制定人员或管理人员要安排一次评审会议，与主要的团队成员一起详细评审测试计划，以保证它的完整性和可用性，并获得批准继续进行。

6.7 测试计划的组成

系统或验收测试计划是基于需求规约的，它需要一个非常好的结构化开发环境和测试环境。系统测试主要评估整个应用程序的功能和性能，是由多种测试组成的，包括性能测试、易用性测试、压力测试、文档测试、安全性测试、容量测试、可恢复性测试等。验收测试是由用户执行的测试，用以验证应用程序是否能够满足最初的业务目标和系统需求，通常是由系统测试的子集组成的。

95

表6-1前后对照了E.2节的各部分与瀑布式生命周期开发的各阶段。交叉点处的Start（开始）表示建议的开始时间或一项测试活动的正式开始，Refine（细化）表示对前一个生命周期阶段开始的测试活动进行细化，Complete（完成）则表示测试活动已完成的生命周期阶段。

表6-1 系统/验收测试计划的阶段对比

章 节	需求阶段	逻辑设计阶段	物理设计阶段	程序单元设计阶段	编码阶段
1. 引言					
a. 系统描述	Start	Refine	Refine	Complete	
b. 目标	Start	Refine	Refine	Complete	
c. 假设	Start	Refine	Refine	Complete	
d. 风险	Start	Refine	Refine	Complete	
e. 应急准备	Start	Refine	Refine	Complete	
f. 约束条件	Start	Refine	Refine	Complete	
g. 批准签字	Start	Refine	Refine	Complete	
2. 测试方法与策略					
a. 测试范围	Start	Refine	Refine	Complete	
b. 测试方法	Start	Refine	Refine	Complete	
c. 测试类型	Start	Refine	Refine	Complete	
d. 后勤	Start	Refine	Refine	Complete	
e. 回归测试的策略	Start	Refine	Refine	Complete	
f. 测试设备		Start	Refine	Complete	
g. 测试规程		Start	Refine	Complete	
h. 测试组织		Start	Refine	Complete	
i. 测试库		Start	Refine	Complete	
j. 测试工具		Start	Refine	Complete	

(续)

章 节	需求阶段	逻辑设计阶段	物理设计阶段	程序单元 设计阶段	编码阶段
k. 版本控制		Start	Refine	Complete	
l. 配置构建		Start	Refine	Complete	
m. 变更控制		Start	Refine	Complete	
3. 测试执行设置					
a. 系统测试规程		Start	Refine	Complete	
b. 设备		Start	Refine	Complete	
c. 资源		Start	Refine	Complete	
d. 工具计划		Start	Refine	Complete	
e. 测试组织		Start	Refine	Complete	
4. 测试规约					
a. 功能分解	Start	Refine	Refine	Complete	
b. 不需要测试的功能	Start	Refine	Refine	Complete	
c. 单元测试用例				Start	Complete
d. 集成测试用例			Start	Complete	
e. 系统测试用例		Start	Refine	Complete	
f. 验收测试用例	Start	Refine	Refine	Complete	
5. 测试规程					
a. 测试用例、脚本、数据开发	Start	Refine	Refine	Refine	Complete
b. 测试执行	Start	Refine	Refine	Refine	Complete
c. 修正	Start	Refine	Refine	Refine	Complete
d. 版本控制	Start	Refine	Refine	Refine	Complete
e. 维护测试库	Start	Refine	Refine	Refine	Complete
f. 自动化测试工具的使用	Start	Refine	Refine	Refine	Complete
g. 项目管理	Start	Refine	Refine	Refine	Complete
h. 监控及状态报告	Start	Refine	Refine	Refine	Complete
6. 测试工具					
a. 将要使用的工具		Start	Refine	Complete	
b. 安装和设置		Start	Refine	Complete	
c. 支持和帮助		Start	Refine	Complete	
7. 人力资源					
a. 所需技能	Start	Refine	Refine	Complete	
b. 角色和职责	Start	Refine	Refine	Complete	
c. 数量和时间需求	Start	Refine	Refine	Complete	
d. 培训需求	Start	Refine	Refine	Complete	

(续)

章 节	需求阶段	逻辑设计阶段	物理设计阶段	程序单元 设计阶段	编码阶段
8. 测试进度安排					
a. 制定测试计划		Start	Refine	Refine	Complete
b. 设计测试用例		Start	Refine	Refine	Complete
c. 创建测试用例		Start	Refine	Refine	Complete
d. 执行测试用例		Start	Refine	Refine	Complete
e. 问题报告		Start	Refine	Refine	Complete
f. 开发测试总结报告		Start	Refine	Refine	Complete
g. 编写测试总结报告		Start	Refine	Refine	Complete

6.8 将技术评审作为持续改进过程

质量控制是质量保证的关键预防性组成部分。在开发生命周期中通过技术评审来排除缺陷就是质量控制技术的一个例子。质量评审的目的是提高开发生命周期的效率，并提供一种方法来度量产品的质量。技术评审减少了返工、测试及“质量逃逸”的数量（“质量逃逸”是指没被发现的缺陷）。技术评审是所缺的排除缺陷的环节，也可以看做一种测试技术，我们甚至可以把测试归类为独立的质量保证组成部分。

审查有好几个别名，而最早的定义是由IBM的Michael Fagan在20世纪70年代提出的。“同行评审”、“审查”、“结构化走查”这几个词通常可以互相替换，意思相同。在开发生命周期的每一个阶段都要执行审查，从用户需求一直到编码。后面还要执行代码走查，开发人员会为评审人员讲解一遍代码。

研究表明，技术评审比执行和测试应用程序所用的自动化测试技术更有效。技术评审是测试的一种形式，或称为手工测试，是不需要在计算机上执行程序的。结构化走查和审查是比单独的软件测试更有效的一种排除缺陷的方法。这些方法还能在生命周期的早期排除缺陷，从而极大地降低排除缺陷的成本。它们代表了一种高效、低成本的排除缺陷的技术，与动态软件测试相比，能潜在地把排除缺陷的成本降低三分之二以上。审查的一个附带好处就是能够定期分析记录的缺陷，并在软件开发生命周期的早期去除导致缺陷发生的根本原因。

下面几节的目的是要为实施软件评审提供一个框架，主要讨论了评审的基本原理、参与者的角色、计划有效评审的步骤、进度、分工、议程定义及评审报告等。

6.9 技术评审的动机

技术评审的动机在于测试所有的软件是不可能的。很显然，没有遗漏的代码测试是不切实际的。测试一个规约或概要设计的技术也是不存在的。“测试”软件测试计划的想法也是令人困惑的。测试也不能解决质量问题或符合标准的程序，而这些对于评审过程则是可能的。

有多种软件技术评审可用于项目，采用哪种主要取决于软件产品的类型及影响评审过程的标

准。评审类型取决于要开发的交付物。例如，对于国防部的合同，就有很多必须遵守的、严格的评审标准。这些需求在公司内部的应用程序开发中可能并不需要。

评审提高了软件产品的质量，减少了返工和不明的工作量，减少了测试的工作量并定义了测试的参数，是一个可重复的、可预见的过程。评审也是发现缺陷和差异的一种有效方法，增加了交付产品的可靠性，对进度有着积极的影响，并能降低开发成本。

早发现错误能减少后面开发阶段的返工，能澄清需求和设计，并确定接口，还能减少测试失败的次数，减少重复测试的次数，确定需求的易测性，并能帮助确认遗漏或有歧义的需求。

6.10 评审的类型

评审可分为正式和非正式的。非正式评审是在同级的员工之间自发进行的，评审者不承担任何责任，并且不必撰写评审报告。正式评审则需召开经过周密计划的会议，评审者要对他们的参与承担责任，并要产生包含行动条目的评审报告。

评审的范围从非常不正式的同行评审到极其正式的结构化审查。评审的复杂性通常与项目的复杂性有关。随着项目复杂性的增加，将需要增加更正式评审。

6.10.1 结构化走查

结构化走查是一种展示性评审，评审的参加者通常就是被评审软件的开发人员，他们讲解软件，然后组内的其他人员在讲解的过程中提出反馈意见。像测试计划、测试用例和测试脚本这样的测试交付物也可以用走查技术进行评审，把它们看做展示性评审是因为大部分反馈通常都仅仅针对实际展示的材料。

评审者并不需要做非常充分的准备。结构化走查的一个潜在不足是由于它是非正式结构化的，所以可能会导致评审无组织、不可控。当开发人员进行走查时，走查工作还可能变得非常紧迫。

6.10.2 审查

审查技术是一个正式定义的过程，用来在整个开发过程中验证软件产品。在从需求到编码的每一个预先定义的阶段都要对所有的软件交付物进行检查，以评定当前的状态和质量有效性。在审查过程中需要决定软件交付物是否符合进入下一个开发阶段的条件。

应该在早期就达到产品的软件质量，因为这时修补缺陷所花的成本是测试或维护阶段成本的1/10到1/100。因此，尽可能早地在软件生命周期中发现并更正缺陷是非常有利的。退出标准是指每个阶段快要结束时通过审查来衡量产品是否完成并达到了所依据的标准。

审查的优势在于非常系统化、可控，并且没有太大压力。审查过程促进了“无自我程序设计”的概念。如果管理得当，可以成为一个论坛，在论坛上开发人员不需对完成的工作持有防护情绪。审查需要一个议程表来指导评审准备和会议本身。对于项目的工作交付物，审查有着严格的进入和退出需求。

结构化走查和审查的主要不同在于，审查通过收集信息来提高开发和评审过程本身。从这种

意义上讲，与走查比起来，审查是一种更好的质量保证技术。

分阶段的审查适用于PDCA（计划、执行、检查、改进）质量模型。每一个开发阶段都有进入的需求。例如，怎样认定进入审查的资格及退出的标准，怎样知道何时退出审查等。在进入和退出之间就是要审查的项目交付物。表6-2列出了分阶段审查的步骤及相应的PDCA步骤。

表6-2 PDCA过程与审查步骤

审查步骤	描 述	计 划	执 行	检 查	改 进
1. 计划	确定参加人员、收集材料、制定宣传演讲日程	√			
2. 宣传演讲	为审查进行的宣传演讲	√			
3. 准备	为审查进行的单独准备		√		
4. 审查	鉴别缺陷的实际审查		√	√	
5. 返工	返工以更正缺陷				√
6. 跟踪	重复前面的工作，以确保修正所有的缺陷				√

持续改进过程的“计划”步骤是由审查计划和准备宣传演讲组成的。审查的策略是要设计和实施一个评审过程，并且是及时的、高效率的、有效的。指定特定的产品，作为可接受的标准，并定义有意义的度量标准来衡量过程的效率，并使其最大化。审查材料必须满足审查的进入标准。要确定合适的参加人员，并要安排日程。另外，还要安排合适的会议地点及时间。应该对参加小组宣讲要审查的内容及他们的角色。

102

“执行”步骤包括为审查单独进行的准备及审查过程本身。参加人员要学习审查材料，按照给他们分配的角色进行准备，并实际执行审查。评审主要从技术精度、标准和惯例、质量保证及可读性几个方面进行，每次评审都会针对产品的上述一个或多个方面进行安排。

“检查”步骤包括对已发现缺陷的确认和文档化。在审查过程中会发现缺陷，但并不会去寻求解决方案或讨论替代的设计方案。审查是一个评审的过程，不是用来讨论解决方案的。

“改进”步骤包括改正缺陷所需的返工和后续跟踪工作。被审查的产品的作者返工改掉所有已经发现的缺陷。团队确保所有的潜在修改措施都是有效的，并且不会由于疏忽而造成二次缺陷。

通过在每个开发阶段都使用审查手段执行PDCA循环，我们验证并改进每个阶段刚刚产生的交付成果，并在发现缺陷时停止这个过程（见图6-4）。在修复所有发现的缺陷之前不能开始下一个阶段。这样做的原因是在越接近缺陷起源的地方发现和修改缺陷就越有利。PDCA过程的重复应用导致了一个上升螺旋，在每个阶段都促进了质量的提高。这样最终的产品质量就会有大幅提高，软件测试过程中令人迷惑的任务也降至最低了；举例来说，在测试团队拿到代码的时候，很多缺陷已经被识别并修改过了。

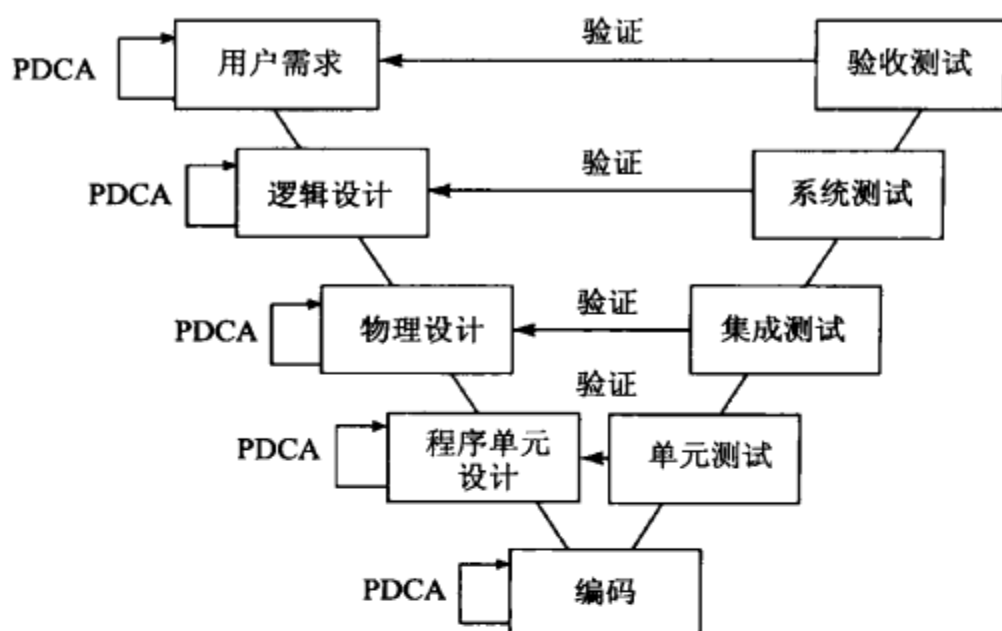


图6-4 作为一个上升螺旋的阶段审查

6.11 参与人员角色

角色取决于使用的具体评审方法，即结构化走查或审查。这些角色都是从职责上说的，这就意味着在一些评审中有可能一个人担任多个角色。评审参与者在评审之后的角色是十分重要的，因为很多在评审中发现的缺陷都没有被开发人员正确地修复。这就产生了一个问题：谁应当对评审进行跟踪或者是否需要另外一次评审。

103

评审主管要对评审负责。这个角色要负责评审的安排，管理有秩序的评审会议，并且准备评审报告。评审主管同时也要保证评审过程的后续工作被恰当的执行。评审主管必须同时拥有技术和人际关系管理两方面的能力。人际关系管理能力包括领导能力、协调技巧和组织天分。评审主管必须保证团队的精力始终集中并且不能让会议成为一个问题解决会议。准备评审的材料不应花费评审主管两个小时以上的时间。

评审过程的记录人员角色确保精确的评审报告所需的所有信息都记录了下来。记录人员必须对复杂的讨论进行分类整理并且抓住行动条目的精髓。记录人员这个角色明显是一个技术职责，不能由非技术人员担任。

评审人员这个角色要客观地对软件进行分析并且对评审负责。一个重要的指导方针就是评审人员必须时刻保持这样的思想：我们是在评审软件，而不是在评审软件的开发人员。这种思想怎么强调都不过分。另外，评审人员的人数不应超过6人，过多的评审人员会导致生产效率比较低下。

在一次技术评审中，开发人员事实上可能会将会议引导为一个对软件的有组织的讨论。需要进行一定程度的准备和计划工作以提供适当的资料。开发人员的态度也很重要，他不采取消极态度是最基本的保证。这可以由团队主管来协调，他要强调审查的目的在于发现缺陷并生产出最好的产品。

104

6.12 有效评审的步骤

6.12.1 步骤 1：规划评审过程

在组织层次和具体的评审层次都要描述计划。组织层次要考虑的是项目所需要的评审的数量和类型。为执行这些评审，必须分配项目资源。

在具体评审层次，计划要考虑的是选择参与人员并定义他们的角色，安排评审时间并且制定评审议程。在选择参与人员的过程中有很多问题，这一般是由管理者通过技术输入完成的复杂任务。选择评审参与人员的时候，必须注意保证评审软件的每个方面都至少分配给了评审团队的一部分人员。

为了将评审过程中的压力和可能的冲突最小化，讨论评审人员在组织中的角色和评审的目的是十分重要的。集中精力于评审目标能够减少人员冲突。

6.12.2 步骤 2：安排评审进度

一次理想的评审应当在开发人员完成软件且其他针对该软件的附加工作还没有开始之前进行。评审主管必须根据设计好的计划安排议程。如果所有的审查条目都没有完成，应当安排另一次审查。

为评审工作分配充分时间的问题在于，估计执行评审所需时间是很困难的。我们使用的方法和为其他会议分配时间的方法是一样的；也就是说，议程需要固定，每个议程条目需要估算时间。一种有效的方法就是在一条时间线上对每个审查条目进行评估。

另一个安排进度上的问题是，评审持续时间过长。这就需要评审过程集中精力于他们的目标。评审参与人员必须在进行评审之前首先了解评审目标以及所暗示的实际评审时间和准备时间。在评审进度排定之前，要评审的交付物首先要满足一些特定条目的进入需求，同时也要定义好退出需求。

6.12.3 步骤 3：制定评审议程

在评审开始之前，评审主管和开发人员首先要制定一个评审议程。尽管评审议程会随着不同的特定产品和评审目标而不同，还是应该为相关的产品类型建立通用的议程。这些议程可以是检查表（参见附录F）。

6.12.4 步骤 4：创建评审报告

评审的一个输出就是报告。报告的格式并不重要。内容应当包括管理者观点、用户观点、开发人员观点和质量保证人员观点。

从管理者的角度来看，评审报告是一个对评审工作的摘要总结，着重于评审了什么，谁进行的评审，以及他们的评价。管理者还需要评估何时能够解决所有行动条目以成功跟踪项目。

用户出于和管理者同样的原因可能关心评审报告的分析。用户可能还希望检验中间产品的质量，作为对开发组织过程的一种监控。

从开发人员的角度来看，最重要的信息包含在行动条目中。这些可能与实际的错误、可能的问题、不一致性或者其他一些开发人员必须考虑的问题相关。

质量保证人员对待评审报告的角度是双重的：质量保证人员必须保证评审报告中记录了所有的行动条目，并且还要关心对评审表单的数据分析和缺陷分类，以改进软件开发过程和评审过程。举例来说，大量的规约错误可能意味着项目的需求规约阶段存在严格程度不够或者时间缺乏等问题。另外一个例子是发现大量缺陷的报告说明了软件没有进行充分的单元测试。

106

测试过程应该在应用程序开发生命周期的早期就开始，而不是像传统的做法，在编码后期才开始进行测试。测试应该与应用程序开发过程紧密地结合在一起。

在软件开发生命周期的需求阶段，业务需求通常是在一个比较高的层面上定义，并且是后续阶段以及最终实施的一个基础。从广义上讲，测试应该在需求阶段开始进行（见图7-1），这样才能够提高系统的质量，以达到用户的期望值。经过测试验证的需求应该是正确的、完整的，但是，事实上通常会产生很多糟糕的需求，从而使开发成本的增加。在瀑布式开发环境中，这些糟糕的需求会影响后面的阶段，最终会导致产品不能满足用户的期望。下面是一些很糟糕的需求具有的某些特性。

- 定义的功能集不完整。
- 没有考虑性能。
- 模棱两可的需求。
- 没有定义安全性。
- 界面没有文档化。
- 需求错误和冗余。
- 需求过于苛刻。
- 需求矛盾。

107

软件功能是整个软件规约中最重要的部分，应该包含层次式的功能分解，这样做能让所有阅读规约的人足够详细地了解软件的层次结构，更重要的是，这样做能够更容易地把规约转化成测试需求。

需求规约的另一个重要元素是数据描述（详见附录C），数据描述应该包含很多细节，诸如数据库究竟是关系型的还是层次型的，如果是层次型，在讲到实体、属性和关系时，就应该用数据模型或实体关系图加以描述。

另外，需求规约应该描述系统和外部实体之间的接口，这些外部实体包括用户、外部软件及外部硬件等。人机交互部分应该描述出用户怎样与系统进行交互，包括界面的形式和用户必须具备的技术能力等。

在需求阶段，测试队伍需要同时完成两项工作，即制定系统/验收测试计划和需求验证。需求验证工作就是要保证开发团队所提交的文档的正确性和完整性。

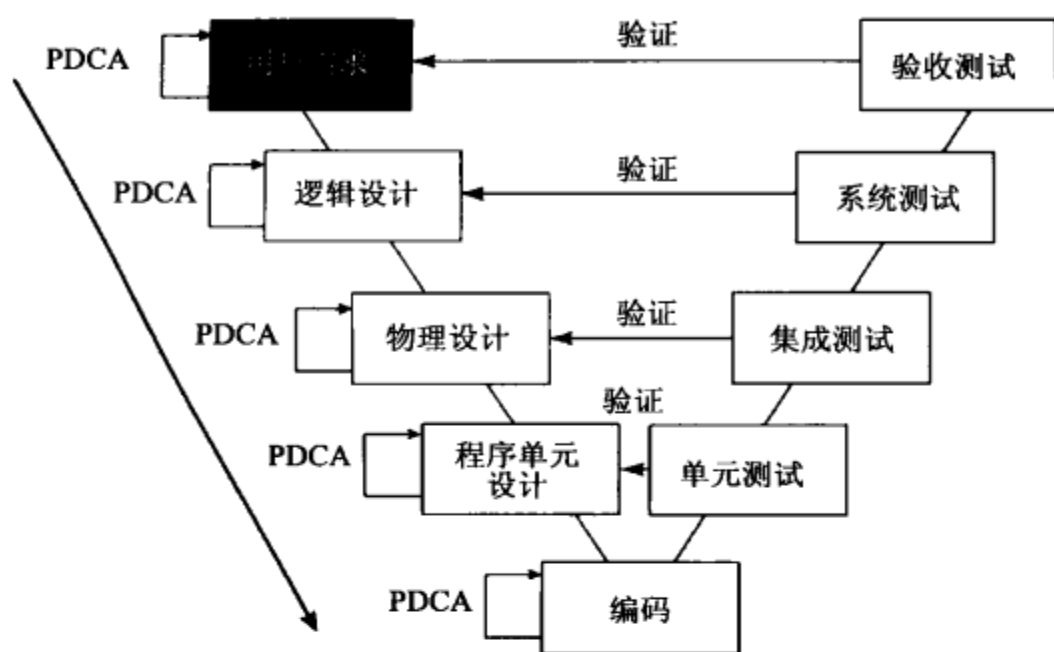


图7-1 需求阶段与验收测试

7.1 通过歧义性评审测试需求

由Bender RBT公司的Richard Bender开发的歧义性评审是一种非常强大的测试技术，它可以在软件测试生命周期的需求阶段消除缺陷，从而避免这些缺陷传播到软件开发生命周期的其他阶段。歧义性评审由接受过这一技术培训的质量保证工程师完成。质量保证工程师不是领域专家（SME），也不阅读需求的内容，只是找出用词的逻辑和结构上的歧义。歧义性评审发生在需求（或者需求部分）得到第一稿之后，以及需求内容（即正确性和完整性）由领域专家被评审之前。质量保证工程师找出需求副本中的所有有歧义的词和短语。结果总结提交给业务分析师。

108

歧义性评审检查表标识编写需求时出现的15种常见问题。

7.2 通过技术评审测试需求

软件技术评审是同行评审的一种形式，在这种评审中，质量团队的人员检查软件产品对目标用途的适用性，并找出与规约和标准的偏离。技术评审还可以提供一些可选方法的建议并对各种可选方法进行检验。技术评审与软件走查不同，它重点关注产品评审的技术质量。它与软件审查不同，它能直接为被评审的产品推荐可选方案，其缺点是直接关注培训和过程改进（摘自Std.1028-1997，即IEEE软件评审标准，3.7款）。

7.3 审查和走查

审查和走查都是一些正式的技术，用来评估文档格式、接口需求，以及前面章节所描述的解决方案约束条件等。

7.4 检查表

列出检查表（表7-1）是为了进行质量控制，检查表中包括为保证需求完整性而提出的一些问题。

表7-1 需求阶段错误记录

错误类别	遗漏	错误	多余	总计
1. 操作规则（或信息）不充分或部分遗漏				
2. 性能标准（或信息）不充分或部分遗漏				
3. 环境信息不充分或部分遗漏				
4. 系统任务信息不充分或部分遗漏				
5. 需求不兼容				
6. 需求不完整				
7. 需求有遗漏				
8. 需求不正确				
9. 规定的精确度不符合实际需要				
10. 数据环境描述不充分				

方法检查表

为了后续更好地讲解需求评审方法，下面介绍一些需求评审方法的步骤和任务。

109 如果评审完全成功，没有发现存在明显的问题，就应该冻结需求规约，任何进一步的修改都必须受到严格的监控。如果评审没有完全成功，在评审过程中发现了一些小问题，就应该由开发人员进行修正，并只由协调人员和相关人员重新对这一部分进行评审。如果在需求评审阶段发现了很严重的问题，那么在错误被修正后，必须由原来的评审团队重新进行一次评审。

需求阶段评审发现的每一个缺陷都应该记录下来。为了正确记录检测出的问题，我们设计了一种需求缺陷报表，包括缺陷分类和缺陷类型。每个缺陷都从遗漏、错误和多余三个方面进行描述，这些缺陷的总结和汇总就是需求评审的结论。表7-1所示就是需求阶段的一部分缺陷记录（详见F.1节）。

7.5 需求可追溯性矩阵

110 需求可追溯性矩阵是一个从分析到实现阶段跟踪用户需求的文档。可以将其用于完整性检查，以验证所有的需求都存在，或者没有不必要的/多余的功能，并且可以将其作为新职员的维护指南。在开发周期的每一步骤中记录需求、代码及相应的测试用例，以保证最终系统能够满足用户需求。用户和开发人员都可以很容易地将需求与设计规约、程序及测试用例进行前后参照。详细信息请参见E.3节。

7.6 制定系统/验收测试计划

验收测试就是要验证系统是否能满足用户的验收标准。验收测试计划是基于需求规约的，并

且要求在正式的测试环境中使用。该测试应用黑盒技术、基于规约来测试系统，通常是由最终用户执行的。在验收测试的过程中，对项目团队来说很重要的一点在于根据实际需要来调整测试过程和更新验收标准。验收测试经常与系统级的测试计划结合在一起，我们现在讨论的就是这种情况。

需求阶段是开发周期中的第一个阶段，这一阶段的工作要在逻辑设计、物理设计、程序单元设计和编码阶段之前完成。在需求阶段，因为还没有足够的信息，所以不要期望测试计划中的所有部分都能完成。

在测试计划的“引言”部分（参见E.2节），一开始介绍的是粗略的测试活动，包括总体的系统描述、验收测试目标、假设、风险、应急准备和约束条件等，这时就要开始为批准签字考虑适当的权限了。

“测试方法与策略”部分中的关键部分包括测试范围、测试方法、测试类型、后勤、回归测试的策略等。测试范围定义了测试工作量的大小，例如是测试全部系统还是一部分。有关测试方法的文档介绍了测试设计方法的基础，如黑盒测试、白盒测试、灰盒测试以及增量式集成等。测试类型确定了将在测试范围内执行的测试的类型，如单元测试、集成测试、系统测试或验收测试。因为缺少详细信息，这时还不能详尽地描述系统级测试的类型，但在下一个阶段则可以得到这些信息。后勤部分描述了开发团队、测试团队与其他相关团队之间的工作关系，也定义了一些具体的事项，如测试组怎样获取被测软件，什么时间获取，缺陷怎样记录、修改和验证等。回归策略则用来确定以前曾测试过的系统功能在引入了新的修改后运行是否正确。

测试需求文档的一个主要难点在于，测试人员必须确定问题定义是否已被正确地转化成了需求文档，这就要求预测一下最终的产品，提出哪些内容应该进行测试，从而确定需求确实解决了问题。

[111]

需求/测试矩阵（见图7-2）是“测试规约”部分中介绍的一种非常有用的技术，可以在对系统进行功能分解时帮助测试人员粗略地进行分析、评审及编写文档，该矩阵定义了项目的测试范围，确保列出了需求规约中的每一项需求，同时帮助测试人员确定哪些功能需要进行测试，哪些功能不需要进行测试。

应用需求/测试矩阵能带来以下益处。

- (1) 能够把测试和脚本与需求关联起来。
- (2) 可以很容易地了解评审的状态。
- (3) 贯穿整个开发周期的跟踪机制，不包括需求、测试用例、缺陷链接在内。

图5-2中的需求/测试矩阵列出了每一项需求，并且将需求与验证这些需求的测试用例和测试脚本关联了起来，列在矩阵左边的需求还能帮助定义“测试方法与策略”部分中提到的系统测试的类型。

针对每项需求只设计唯一的测试用例通常是不太可能的。事实上，为了全面地测试一项需求，往往设计多个测试用例，这也使得某些测试用例对其他需求的可复用性成为可能。一旦构建了需求/测试矩阵，就可以开始着手测试用例的设计和测试脚本的创建等工作了。

每个测试用例都与某项需求相关联，状态列是用来跟踪每条测试用例的状态的。例如，状态列中的“Q”就表明需求已经被质量保证人员（QA）评审过了，“U”表明用户已经评审过该需

求，“T”则表明测试用例规约已被评审通过，准备进行测试。

测试需求	测试用例									备 注
	1	2	3	4	5	6	7	8	9	
功能										
1										
2			Q						T	
3										
4		Q						Q		
性能										
1										
2			T							
3		Q					Q		Q	
4										
安全性										
1				U						
2										
3					Q					
4				U					T	

U: 用户已评审过的。
Q: 质量保证人员已评审过的。
T: 准备进行测试的。

图7-2 需求/测试矩阵

在测试计划的“测试规约”部分，可以获得验收测试的信息并记录下来。为了让用户接受这个系统，这些测试是必须通过的。规程是指运用某种操作模式执行一系列的相关活动，即指明怎样来完成某些事情。在“测试规程”部分中可以记录下列信息：测试用例、脚本、数据开发、测试执行、修正、版本控制、维护测试库、自动化测试工具的使用、项目管理、监控及状态报告等。

现在应该开始考虑所需要的测试人力资源了，包括所需技能、角色和职责、数量和时间需求以及人员的培训需求等。

112
113

业务需求是在需求阶段定义的。逻辑设计阶段对业务需求加以优化和完善,为物理设计和编码过程中将会用到的系统规约做必要的准备。从功能模型和信息模型的角度来看,逻辑设计阶段进一步对需求阶段定义的业务需求进行了细化。

8.1 数据模型、过程模型及其联系

逻辑设计阶段为构建应用程序建立了详细的系统框架。这个阶段的3个主要成果是数据模型(即实体关系图)、过程模型以及两者的联系。

数据模型是对应用程序所需信息或数据对象类型的表述。它建立了应用程序中人、地点和重要事件之间的联系,并在接下来的数据库物理设计中使用(数据库物理设计是物理设计阶段的一部分)。数据模型是一种用于定义实体和关系的图形化技术。实体是我们想存储相关数据的事物。它是用户关注的某一可唯一辨识的人、地点、物或事件,应用程序要维护和报告的正是与这些实体相关的数据。实体的一些例子包括客户、订单、办公室和购物订单等。

每个实体都是一个被水平分割成行和列的表。每一行是实体的一个具体出现,非常类似于文件中的记录。每一列则是用于描述实体的一个属性。属性的例子有尺寸、日期、价值和地址等。数据模型中的每个实体都不能独立存在,而是通过关系同其他实体相互关联的。关系是在用户关注的两个或多个实体之间的关联,应用程序所要维护和报告的就是这些实体的数据。有3种关系:一对一的关系,将一个实体的一次出现与另一实体的零次或一次出现相关联;一对多的关系,将一个实体的一个出现与另一实体的零次或多次出现相关联;多对多的关系,将一个实体的多次出现与另一个实体的多次出现相关联。关系类型定义了实体关系的维度。有关数据建模更多的详细信息,请参见G.10节。

过程是与相关联的输入和输出结合在一起的业务活动。过程的例子有接受订单、更新库存、配货订单和课程安排等。过程模型是一种图形表示,应该描述过程做什么,而不是指过程为什么、怎样及何时实现。这些是在物理设计阶段中定义的过程的物理属性。

过程模型是对业务的一种分解方式。过程分解是将活动分解为连续的详细步骤的过程。它自顶向下进行,直到最基本的过程(即对于用户有意义的活动的最小单元)才停止。

过程分解图用于以分层结构描述过程,它显示了更加详细的分层结构。这个图将过程迭代地建立,非基本过程都会得到分解。过程的根结点是分解的开始点。父结点是比较低等级更高的过程,

而子结点则是连接在高一级或父结点上的较低等级的过程。数据流图经常用来验证过程分解,它显示了所有的过程、数据存储的访问和输入输出数据流。它也显示了过程外部流入和流出实体的数据流。

有一种关联图通常叫作CRUD矩阵,或者过程/数据矩阵,将数据模型和过程模型联系起来(见图8-1)。它有助于确保数据和过程的发现和评估。CRUD矩阵标识并解决了矩阵冗长和冲突问题,在必要时协助优化数据模型和过程模型。它将过程映射到实体,显示一个实体中哪些过程创建、读取、更新或删除了实例。

过程 \ 实体	实体类型									备 注
	1	2	3	4	5	6	7	8	9	
计划	crud				cu			cu		
出售		ud	c				c			
日程安排	c				d	crud			d	
赔偿			cu	c	d		cu			
装运		crud	ud	u	c	crud				
操作					crud		crud			
维护		c			cu				cu	
成本计划	crud					crud				
购买			ud					d		
预测							c			
接收		c	c		c					
下单	d					d			cu	
研发			crud		c		crud			
· · ·										

图8-1 CRUD矩阵

用CRUD所做的分析一般称为“实体生命周期分析”,它分析了一个实体的产生和消亡,由实体相关的过程来执行。该分析首先验证了有一个相关的过程在实体中创建了一个实例。如果某个实体没有创建它的相关过程,就说明遗漏了一个过程,必须加以定义。接下来,该分析验证了存在相关过程来更新、读取或删除一个实体中的实例。如果有一个实体从未被更新、读取或删除过,说明这个实体可能已经被清除了。关于软件测试中如何应用这一分析的详细信息请参见G9节。

8.2 通过技术评审测试逻辑设计

逻辑设计阶段是用静态技术加以验证的,也就是说并不执行应用程序。像在需求阶段中使用

的那样，这些静态技术检查对规约规定的遵循程度和模型的完备性。逻辑设计阶段使用的是与验证需求时所用的一样的静态测试。需要加以评审的工作成果包括数据模型、过程模型和CRUD矩阵。

在逻辑设计评审中发现的每个缺陷都应记录在案。可以设计一份缺陷问题报告对这些缺陷进行恰当的记录，包括缺陷种类和缺陷类型。对每个缺陷的描述都记录在遗漏、错误或额外列中。作为对逻辑设计评审的结论，应对缺陷加以总结并汇总。表8-1给出了一个逻辑设计阶段缺陷记录表单的例子（更多细节参见F.2节）。

表8-1 逻辑设计阶段缺陷记录

缺陷种类	遗 漏	错 误	额 外	总 数
1. 数据定义不充分				
2. 实体定义不完整				
3. 实体基数不正确				
4. 实体属性不完整				
5. 违反规范化要求				
6. 不正确的主键				
7. 不正确的外键				
8. 不正确的组合键				
9. 不正确的实体子类型				
10. 进程定义不充分				

8.3 细化系统/验收测试计划

系统测试是一种多方面的测试，对整个应用程序的功能、性能和适应程度加以评估。系统测试可以证明系统是否满足初始目标。需求阶段无法得到足以定义这些测试类型的详细信息，而逻辑设计则提供了数据模型和过程模型的详细信息。现在我们可以对测试步骤和策略环节中的测试范围和测试类型加以细化，使之包括即将进行的系统级别测试类型的相关细节（参见E.2节）。系统级测试的例子包括功能测试、性能测试、安全性测试、易用性测试和兼容性测试等，用于衡量系统适于使用的程度。在这一部分中细化了测试方法、逻辑和回归策略，并且开始论述本部分的其他议题，例如测试设备、测试规程、测试组织、测试库和测试工具等。软件配置管理要素的初步计划也开始描述，例如版本控制和变更控制以及配置构建等。如果在组织中还没有配置管理工具，计划中还包括获取它。

“测试执行设置”部分处理那些为测试做准备的注意事项，包括系统测试过程、测试设备、需要的测试资源、测试工具计划和测试组织等。

在“测试规约”部分中，我们可以从数据模型和过程模型中得到更多的功能细节，它们将被添加到需求/测试矩阵。这时，可以开始系统级测试用例的设计。但是要完成详细的测试开发（例如，与每个测试用例相关的测试过程、测试脚本和测试用例输入/输出数据）还为时过早。在这个阶段中应完成验收测试用例。

在“测试规程”部分中，细化了前一阶段开始进行的项目。下面开始论述“测试工具”和“测试进度安排”部分中的测试项目。

逻辑设计阶段将业务需求翻译成可在物理设计和编码过程中由程序员使用的系统规约。物理设计阶段决定了需求自动化的方法。在这个阶段中创建了概要设计，在该设计中定义了基本的过程组件、它们的相互关系和主要的数据表示等内容。

物理设计阶段建立了系统的架构，或者说是结构方面。逻辑设计测试是功能性的，而物理设计测试则是结构性的。这个阶段验证了设计的结构完整性以及是否能够实现已经文档化的需求的意向。在这个阶段我们假设需求和逻辑设计是正确无误的，并把注意力放在设计的完整性上。

9.1 通过技术评审测试物理设计

逻辑设计阶段是用静态技术加以验证的，也就是说并不执行应用程序本身。在需求阶段和逻辑设计阶段，这些静态技术通过关注架构设计来检查对规约规定的遵循程度和完备性。物理设计验证的基础是用于描述设计的设计表示方案。设计表示方案的例子包括结构图、Warnier-Orr图、Jackson图、数据导航图和已在逻辑设计阶段中绘制的关系数据库图。

121

设计表示方法提供了一种机制，可以描述算法及其对软件模块的输入和输出。在描述模块中数据对象的控制流时可能出现各种各样的问题。例如，一个模块可能需要另一模块创建的某个特定的数据项，但是这个数据项却没有正确的给出。可以应用静态分析方法来检测这类控制流错误。

物理设计中产生的其他错误也是可以检测的。通过迭代地给出详细信息，我们可以编写设计规约。尽管分层次的规约结构是十分有效地阐述整个设计的工具，但它不考虑细节级别之间存在的矛盾。例如，耦合度体现了模块之间的独立性。当两个模块之间的交互很少时，模块之间是松散耦合的；当交互很多时则为紧密耦合。一般认为松散耦合是一种好的设计实践方法。

耦合的例子包括内容耦合、共用体耦合、控制耦合、时间戳耦合和数据耦合。内容耦合发生在一个模块引用或修改了另一个模块的内在内容时。数据耦合发生在两个模块通过一个变量或数组（表）进行通信时，这些变量或数组（表）作为参数在两个模块间直接传递。静态分析技术可以检查出是否出现耦合。

设计表示的静态分析能够检测静态错误和语义错误。语义错误涉及信息分解或数据分解、功能分解和控制流。在物理设计评审中发现的每一个缺陷都应当记录在文档中，并加以分类、记录、提交给设计团队进行修改，并在记录该缺陷的特定文档中提及。表9-1是物理设计阶段缺陷记录表单的一个例子（更多细节参见F.3节）。

表9-1 物理设计阶段缺陷记录

缺陷种类	遗 漏	错 误	额 外	总 数
1. 逻辑或次序有错误				
2. 处理不准确				
3. 例程没有输入或输出所需的参数				
4. 例程没有接受在容许范围内的所有数据				
5. 对输入数据进行限制和有效性检查				
6. 恢复流程没有实现或不充分				
7. 所需进程缺失或不充分				
8. 值有错误的或不明确				
9. 数据存储有错误或不充分				
10. 变量缺失				

9.2 创建集成测试用例

集成测试是用来对软件的结构和架构进行测试的，它判定是否所有的软件组件都被恰当地连接。集成测试不负责验证系统功能的正确性，只负责验证系统按照设计完成。

集成测试是识别由于将经过单个程序单元测试的模块组合到一起时而引入的错误的过程。只有当所有的程序单元都能够按照单元规约的要求运行时，才能开始集成测试。集成测试可以先把几个逻辑单元集成起来开始测试，也可以把所有单元一次集成起来。

因为集成测试的着眼点是各个单元之间的正常连接，所以这个测试的目的是保证所有程序单元能够集成到一起，参数能够传递，以及文件能够正确处理。集成测试技术包括自顶向下、自底向上、三明治测试和线索测试（更多细节参见附录G）。

9.3 集成测试方法

接下来描述创建集成测试用例的一种方法。

9.3.1 步骤 1：标识出单元接口

每个程序单元的开发人员将该单元的接口标识出来并记录在文档中，用于下列单元操作。

- ☐ 外部查询（对终端所查询信息的反馈）。
- ☐ 外部输入（对输入待处理事务数据的管理）。
- ☐ 外部文件操作（对计算机文件进行获取、更新或创建事务的操作）。
- ☐ 内部文件操作（传递或接收来自于其他逻辑处理单元的信息）。
- ☐ 外部显示（向终端发送消息）。
- ☐ 外部输出（向某个输出设备或单元提供处理结果）。

9.3.2 步骤 2：全面协调接口

应该为被测软件中所有程序单元收集集成测试模板所需的信息。只要一个单元与其他单元关

联,就需要协调关联的接口。例如,如果程序单元A向程序单元B传输数据,程序单元B应表明它已经接收到程序单元A的输入。在执行集成测试前,应该检查未协调的接口。

9.3.3 步骤3: 创建集成测试条件

在集成每个程序单元之前需准备好一个或多个测试条件。创建测试条件之后,应将测试条件的数目记录在测试模板中。

9.3.4 步骤4: 评估集成测试条件的完整性

下面的问题列表将帮助评估集成测试条件的完备性,这些测试条件记录在集成测试模板中。这个列表也可帮助判定集成过程创建的测试条件是否完备。

- ☐ 是否为下面的每个外部查询都相应开发了一个集成测试?
 - 记录测试
 - 文件测试
 - 搜索测试
 - 匹配/合并测试
 - 属性测试
 - 压力测试
 - 控制测试
- ☐ 是否模块之间的所有接口都进行了验证,以使一个模块的输出可以作为另一个模块的输入?
- ☐ 是否开发了文件测试事务? 模块是否与指定的文件进行关联?
- ☐ 在集成测试之前,是否所有的单元处理过程都得到了有效验证?
- ☐ 是否所有的单元开发者都认为集成测试条件已经足够测试每个单元的接口?
- ☐ 在集成测试中是否包括了所有的软件单元?
- ☐ 在集成测试中是否包括了当前被测软件使用的所有文件?
- ☐ 在集成测试中是否包括了与当前被测软件相关的所有业务事务?
- ☐ 在集成测试中是否包括了当前被测软件包含的所有终端功能?

集成测试的文档工作是在“测试规约”部分开始着手进行的(参见E.2节)。在这部分中,虽然还会继续细化功能分解,但系统级测试用例应该在该阶段完成。

在这个阶段中还要完成“引言”部分中的测试项目,并对“测试方法与策略”、“测试执行设置”、“测试规程”、“测试工具”、“人力资源”和“测试进度安排”等部分继续加以细化。

124

125

设计阶段开发了系统的物理架构或结构方面。为了进行更加详细的设计，要对程序单元设计阶段加以细化。程序单元设计是详细设计，其中确定特定的算法和数据结构。程序单元设计也是对详细控制流的阐述，使之很容易用一种编程语言翻译成程序代码。

10.1 通过技术评审测试程序单元设计

一个好的详细程序单元设计应当可以很容易地翻译成多种编程语言。详细程序单元设计会使用一些结构化的技术，例如while、for、repeat、if和case结构等，这些都是在结构化编程中使用的结构。结构化编程的目的是以较低的成本开发出高质量的程序。结构化程序中应当只使用三种最基本的控制结构。

10.1.1 顺序结构

语句按照源代码列出来的顺序一条接一条地执行。赋值语句就是一个顺序结构的例子。

10.1.2 选择结构

对条件进行测试，根据测试结果的是与否，决定执行一条或多条不同的路径。if-then-else结构是选择结构的一个例子。在这个结构中，对条件加以测试：如果条件为真，执行一组指令；如果条件为假，则执行另一组指令。两组指令连接在一个公共点上。

10.1.3 循环结构

循环结构用于多次循环执行一组指令。循环结构的例子包括do-until和do-while。do-until循环执行一组指令，然后测试循环结束条件：如果条件为真则循环结束，继续下面的结构；如果条件为假则再次执行这一组指令，直到满足结束逻辑条件为止。do-while循环首先测试结束条件：如果为真则控制被传递到下面的结构；如果为假则执行一组指令，直到控制被无条件地传递回条件逻辑。

对详细设计的静态分析可以检测出涉及信息和逻辑控制流的语义错误。程序单元设计评审中发现的每个缺陷都应当记录在文档中，并加以分类、记录、提交给设计团队进行修改，并在记录该缺陷的特定文档中提及。表10-1是程序单元设计阶段错误记录表单的一个例子（更多细节参见

F.4节)。

表10-1 程序单元设计阶段缺陷记录

缺陷分类	遗漏	错误	额外	总数
1. “if-then-else”结构的使用不正确?				
2. “do-while”结构的使用不正确?				
3. “do-until”结构的使用不正确?				
4. “case”结构的使用不正确?				
5. 存在死循环吗?				
6. 程序正常吗?				
7. 有“goto”语句吗?				
8. 程序可读吗?				
9. 程序高效吗?				
10. “case”结构能覆盖所有条件吗?				

10.2 编写单元测试用例

单元测试是执行软件系统一个功能子集的过程，以判定这个子集是否完成了指定的功能。单元测试是面向功能或模块的检查。创建白盒测试用例，并将其记录在文档中，用以验证单元逻辑；创建黑盒测试用例用以根据规约进行测试（一个测试用例表格的例子参见E.8节）。单元测试和在修改与回归验证过程中所需的版本控制一般都是由开发人员执行的。在单元测试用例的开发过程中，了解哪一部分代码被测试覆盖而哪些没有被覆盖是非常重要的。通过了解测试用例的覆盖情况，开发人员可以发现从未执行过的代码或者没有按照规约的描述开发的程序功能。在覆盖不完全的情况下，实现整个系统是有风险的，因为缺陷可能出现在没有测试过的代码中（更多单元测试用例开发技术参见附录G）。“测试规约”部分阐述并记录了单元测试用例说明（参见E.2节），但是该部分的其他项目应当已经全部完成。

“引言”、“测试方法与策略”、“测试执行设置”、“测试工具”和“人力资源”中的所有项目都应在这个阶段之前完成。然而，“测试规程”部分中的项目还要继续加以细化。功能分解、集成测试用例、系统测试用例和验收测试用例都应在这部分完成。“测试规程”和“测试进度安排”部分的所有项目继续加以细化。

128
129

程序单元设计是详细设计，其中确定特定的算法和数据结构。对详细控制流的阐述使之很容易用一种编程语言翻译成程序代码。编码阶段是使用编程语言将详细设计翻译成可执行的代码的过程。

11.1 用技术评审测试编码

编码阶段产生可以执行的源程序模块。编码工作的良好基础是已定义的编程标准。好的标准应当包括注释、不安全的编程结构、程序布局、防御性编程等。注释是指一个程序应当如何被注释以及注释到何种程度。不安全的编程结构是使程序难于维护的习惯，比如goto语句。程序布局指的是一个标准程序应如何在一个页面上布局、控制结构的缩进和初始化等。防御性编程实践描述了程序防御策略的强制元素，一个例子就是错误情况的处理和对常见错误例程的控制。

静态分析技术，如结构化走查和审查，用于确保程序代码及其文档的适当形式。这是通过对代码和文档规则的遵循程度的检查以及类型检查完成的。

编码阶段的评审中发现的每个缺陷都应当记录在文档中，并加以分类、记录、提交给设计团队进行修改，并在记录该缺陷的特定文档中提及。表11-1是编码阶段错误记录表单的例子（更多细节请参见F.5节）。

表11-1 编码阶段缺陷记录

缺陷种类	遗漏	错误	额外	总数
1. 判断逻辑或顺序错误或不充分				
2. 数学计算错误或不充分				
3. 分支错误				
4. 分支或其他测试执行错误				
5. 存在未定义的循环终止条件				
6. 违反编程语言规则				
7. 违反编程标准				
8. 程序员错误解读语言结构				
9. 存在印刷错误				
10. 存在内存分配错误				

11.2 执行测试计划

132

在这个阶段的末尾，测试计划的每部分的所有项目都应已经完成。测试计划在需求阶段、逻辑设计阶段、物理设计阶段和程序单元设计阶段中完成，而软件的实际测试将通过测试计划中的测试数据完成。由于结果已经在测试用例和测试规程中阐述，从静态测试的观点来看，已经保证了程序执行的正确性，也就是说测试已经通过了手工评审。

动态测试，即依赖于时间的测试技术，涉及用电脑执行特定顺序的指令集。这些技术用于研究代码的功能正确性和计算正确性。

动态测试的进行顺序与开发生命周期正好相反。动态测试从单元测试开始，独立验证每个程序单元，然后进行集成测试、系统测试和验收测试。验收测试结束后系统就可以投入运营和维护了。图11-1简单描述了每种测试类型。

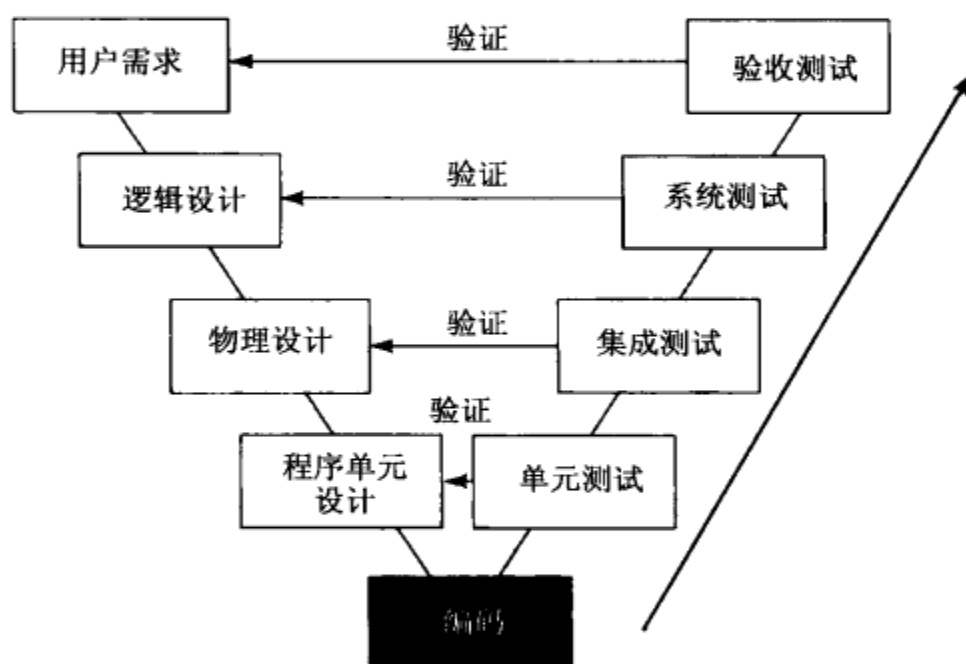


图11-1 执行测试

11.3 单元测试

单元测试是测试的基础级别。单元测试分别着眼于程序或系统的较小构建块，是执行每个模块以证实其实现了指定功能的过程。单元测试的优势在于它允许对小单元的测试和调试，因此为管理从小单元到大单元的集成过程提供了更好的方式。另外，测试小单元的代码可以用更少的测试达到在数学意义上对代码逻辑进行完全测试的目的。单元测试也有利于自动化测试，因为较小单元的行为可以被捕获并回放，具有最大的可复用性。单元可以是应用程序软件的几种类型之一。例子包括模块，其本身即可作为一个单元，图形用户界面组件，如窗口、菜单；函数、批处理程序、联机程序和存储过程等。

133

11.4 集成测试

单元测试结束后必须对所有模块进行集成测试。在集成测试中，通过一次将一个或多个模块加入到已经集成的模块核心，把系统逐渐构建起来。在系统测试之前，单元组已经进行过充分测试。由于在集成测试之前，模块已经过单元测试，可以作为黑盒对待，这使得集成测试可以重点关注模块之间的接口。集成测试的目标是验证在控制结构下每个模块都能正确运行以及验证模块接口的正确性。

通过对模块的逐步组装执行增量测试。在每一步中，向程序结构中添加一个模块，测试重点放在运行这个新增的模块上。当这个模块证明是可以在系统结构中正确运行的时候，再加入一个新的模块，并继续测试。重复这个过程，直到所有的模块都被集成在一起并测试完毕。

11.5 系统测试

在集成测试之后，根据系统/验收测试计划，整个系统作为一个整体其功能性和适于使用的程度进行测试。在验收测试开始之前，需要在计算机操作环境下对系统进行全面测试。系统测试的根据是在软件质量保证计划中阐述的质量属性。系统测试是一组测试，用来验证这些质量属性并保证验收测试开始的时候相对来说已经不存在问题了。系统测试验证系统的功能都能正常运行，以及系统具有某些特定的非功能特性。系统测试的一些例子包括易用性测试、性能测试、压力测试、兼容性测试、转换测试和文档测试等。

黑盒测试是一种着重按照规约测试程序功能的技术。白盒测试则是一种测试逻辑路径以确定得到可预测结果的可靠性的测试技术。灰盒测试是以上两种技术的结合，经常用于进行系统测试。灰盒测试是两种技术折中的产物，也是在系统测试中广泛使用的平衡的测试方法。

11.6 验收测试

在系统测试之后，验收测试保证软件系统满足初始需求。在软件成功完成系统测试以前，不会进行这个测试。验收测试是用户执行的测试，使用黑盒技术对照系统规约对系统进行测试。最终用户负责确保所有相关功能都被测试到。

验收测试计划定义了执行验收测试的步骤，并且在执行过程中应尽可能地按照计划去做。即使在执行过程中找到了错误，验收测试也会继续进行下去，除非该错误已经阻止了验收测试的执行。有些项目并不需要正式的验收测试。比如，当客户/用户由于其他的一些系统测试已经对产品很满意了，或者由于时间的限制，或者最终用户参与了产品开发的整个周期并且在开发过程中已经隐性地执行了验收测试。

验收测试一般来讲是一个或多个的系统测试子集。还有如下两个衡量验收测试的方法。

(1) 平行测试，与现有系统进行一个业务事务层次的对比，目的是保证新的系统提供了足够的结果。

(2) 基准，一系列手工编写或者由当前系统生成的期望结果的静态集，作为对新系统的期望测试结果。

11.7 缺陷记录

在上述测试过程中发现的每个缺陷都要写在文档中，以协助适当地记录这些缺陷。当在测试过程中出现了测试人员无法解释的事件时，要给出一个问题报告。问题报告中记录该事件的详细情况和以下条目（详细信息参见E.12节）。

- 问题标识。
- 作者。
- 发布/构建编号。
- 打开日期。
- 关闭日期。
- 问题域。
- 缺陷或增强。
- 测试环境。
- 缺陷类型。
- 缺陷发现者。
- 缺陷如何发现。
- 缺陷分配给的人。
- 优先级。
- 严重性。
- 状态。

其他用来对测试过程和测试结果进行交流的测试报告包括测试用例日志、测试日志总结报告和系统总结报告等。

135

测试用例日志记录将要执行的某个测试类型的测试用例，同时也记录测试的结果，为测试日志总结报告提供详细的依据，并为重构测试提供可能（如果需要的话）。详细信息参见E.9节。

测试日志总结报告记录测试人员日志中的测试用例，这些尚未完成或者已经完成的测试用例用于状态报告和度量标准收集。详细信息参见E.10节。

系统总结报告应该为每一个主要的测试事件做准备，有时还需要对所有的测试进行总结。下面列出了通常包含的主要部分：一般信息（描述测试目标、测试环境、参考文献），测试结果及发现（描述每个测试），软件功能及发现，以及分析和测试总结。详细信息参见E.11节。

136

螺旋（敏捷）软件测试方法： 计划、执行、检查、改进

螺旋式开发方法是对传统的瀑布式系统开发方法的一种改进，在瀑布式开发方法中，产品开发是分阶段按顺序进行的。生命周期开发模型的一个普遍问题是交付产品会延期，因为用户仅仅在开发过程最开始的阶段和最后的阶段才介入，最终的结果就是交付到用户手上的系统往往不是他们最初想要的。

与之相反，螺旋式开发方法则加速了产品的交付。一个具有一定功能的小系统很快就被构建出来并交付给用户，然后在一系列迭代过程中系统得到增强。这种做法的一个优点是用户可以很快得到一些功能，另一个优点是产品在不断的反馈中逐渐成形。例如，用户不需要在开发周期的开始就必须正确地、详细地定义出每个功能，但是需要用户能够对每次迭代进行反馈。

螺旋测试是动态的，传统意义上的系统交付可能永远也不会完成。“螺旋”这个术语说明了这样一个事实，传统的按顺序的“分析-设计-编码-测试”等各个阶段会在一个短时间的螺旋或者周期里小范围地实现，然后这些阶段在后续的几个周期里重复出现。

137

本部分的目标如下：

- 讨论瀑布式开发的局限性；
- 描述客户/服务器架构的复杂性；
- 讨论螺旋测试的心理学；
- 描述迭代/螺旋开发环境；
- 将Deming的持续性质量改进原理应用于螺旋式开发环境。
 - 信息收集；
 - 测试计划；
 - 测试用例设计；
 - 测试开发；

- 测试执行/评估；
- 可追溯性/覆盖率矩阵；
- 准备下一个螺旋；
- 系统测试；
- 验收测试；
- 总结/报告螺旋测试的测试结果。

12.1 生命周期开发的局限性

在第2章中我们回顾了瀑布式开发方法以及相关的测试活动。生命周期开发方法包含从需求到编码的不同阶段。生命周期测试是指与开发生命周期并行发生的测试，并且是一个持续的过程。尽管生命周期开发或瀑布式开发对于很多大型应用程序是非常有效的，例如，DOD、金融、安全性应用程序等，但是这种方法也有很多缺点。

- 系统的最终用户仅仅在开发过程最开始的阶段和最后的阶段才介入，最终的结果就是交付到用户手上的系统往往不是他们最初想要的。
- 开发周期的延长和业务周期的缩短导致真正需要的软件和交付的软件之间存在差距。
- 在编码之前，要求最终用户详细地描述他们在系统中想要的功能，这对于开发人员而言是符合逻辑的，但是，有些用户此前甚至还没有使用过计算机，他们对计算机的能力并没有多少认识。
- 当开发阶段接近尾声的时候，实际上开发并没有彻底完成，但是流程和项目计划要求无论如何开发工作必须完成。事实上这个阶段根本没有完成，而且需要做的事情总是比能够做的要多。这就导致了“涟漪效应”——项目迟早会返回到这个阶段以真正完成工作。
- 经常发生的情况是，瀑布式开发方法并不能得到严格的执行。在需要快速地产生一些结果时，该方法的一些关键的部分没有得到执行。最坏的情况是随意开发，分析和设计阶段被忽略，而编码阶段成为了首要的活动。这是一个非结构化开发环境的例子。
- 软件测试经常被看做是一个独立阶段，是在编码阶段才开始的验证技术，而没有集成到整个开发生命周期中。
- 瀑布式开发方法对很多开发项目来说是非常不够的，即使是采用了这种方法。一个软件系统如果不是用户想要的，那么即使完成也不会有多大的价值。如果需求没有被完整地记录下来，系统不会通过用户的验收过程，也就是说，这是一个错误的系统。另一种情况是，需求是对的，但设计与需求不一致，最终的系统也是错误的。重申一下，已经完成的产品很可能在系统的验收过程中失败。
- 基于上述原因，专家们开始发表一些基于其他解决方式的开发方法，如原型法。

12.2 客户/服务器架构的挑战

应用程序开发中如果采用客户/服务器架构,就必须注意在客户和服务器两者之间合理分配功能,客户和服务器才可以独立地完成各自的任务。客户与服务器一起协作来产生所需要的结果。

客户是作为一个用户的智能工作站,因为客户拥有自己的操作系统,所以它可以运行其他的应用程序,如电子表格、文字处理软件、文件处理软件等。用户和服务器协同处理客户/服务器架构的一些应用功能。服务器可以是一台PC、小型机、局域网也可以是大型机。服务器接受来自客户的请求并加以处理。硬件的配置由应用程序的功能性需求来确定。

客户/服务器应用程序的优点包括降低了成本、改进了数据的可访问性而且系统灵活性较好。但是,评估客户/服务器方法和保证质量非常困难,而且带来了其他在大型机应用程序中未必出现的困难。问题主要是下面这些。

140

- 典型的图形用户界面有更多可能的逻辑路径,因此在大型机环境中大量的测试用例混杂在一起。
- 客户/服务器技术是比较复杂的,对很多机构来说经常是全新的。此外,这种技术经常来自于多个不同的供应商,并且在多种不同的配置和不同版本的情况下应用。
- 客户/服务器应用程序通常分布广泛,这就导致了大量的错误源的出现,以及硬件/软件配置控制问题的发生等。
- 必须要根据整个机构的成本和收益做一个短期和长期的成本/收益分析,以评判客户/服务器技术。
- 要想成功地把系统移植到客户/服务器架构中,依赖于整个机构对客户/服务器技术的接受程度是否与移植计划相匹配。
- 客户/服务器技术对用户业务的影响可能是非常巨大的。
- 选择哪些应用程序作为实施客户/服务器实现的候选程序,不是那么容易。
- 哪些开发技术和工具能够适用于客户/服务器架构,需要分析。
- 必须衡量客户/服务器架构的技能和资源的可用性,这些成本较高。
- 尽管客户/服务器技术的成本要比大型机系统贵得多,但成本不是唯一的问题,功能、业务收益和来自最终用户的压力等都要去平衡。

客户/服务器环境中的集成测试是很有挑战性的。客户和服务器的应用程序是分别开发的。当它们被集成到一起的时候,无论接口定义得多么清楚都会发生一些冲突。当把应用程序集成到一起的时候,对一些缺陷的解决方案可能有一个,也可能有多个,质量保证组和开发组之间必须有良好的沟通。

在很多场合存在的说法是大型机不灵了,客户/服务器架构要盛行下去。事实上基于大型机架构的应用没有消失,客户/服务器技术也不是所有应用的万能良方。两种架构将继续共存并互相补充。大型机架构肯定会是客户/服务器策略的一部分。

12.3 客户/服务器架构中螺旋测试的心理学

12.3.1 新思想

生命周期测试的心理学倡导由开发组织以外的人员进行测试。之所以这么做是因为在生命周期开发方法中通常具备清晰定义的需求，由第三方来验证需求是更高效的。测试经常被看做是为破坏开发人员工作成果而设计出来的具有破坏性的过程。

141

但是，螺旋测试的心理学则鼓励质量保证组织与开发组织合作。这种论点的出发点是，在快速应用程序开发环境中，需求可能已经准备好，也可能还没有准备好，各种情况都有。如果没有这种合作，测试部门将会很难定义测试的标准。唯一可能的选择是测试组织和开发组织一起协同工作。

测试人员可以是开发团队的强大同盟军，只要稍加努力，他们之间的关系就可以从对手转变成合作伙伴。这完全有可能，因为大多数测试人员是愿意提供帮助的，他们只是需要些关注和支持。为了达到这一点，需要创造出能够让测试人员发挥能力的环境。测试人员和开发经理必须在开发的早期为合作创造条件，并且要在整个开发生命周期中加强沟通。

12.3.2 对测试人员/开发人员的理解

要想理解测试部门和开发部门之间良好关系的一些内因，首先要理解他们是如何看待自己的角色和职责的。

测试是一项艰苦的工作，是一项不可能穷尽的、不确定的任务。无论测试人员做到什么程度，他们都无法确定自己是否发现了所有的问题，甚至是连重要的问题是否都发现了也无法保证。

很多测试人员并不是真正地对测试感兴趣，也没有在基本的测试原则和技术方面接受正规的培训。一些测试方面的书籍或会议通常把测试的主题搞得很晦涩，并且使用一些很深奥的数学分析。要求有正规的需求规约作为有效测试的先决条件，在实际的软件开发项目中往往是不现实的。

找到擅长测试的人才是很难的。测试人员必须具备下列素质：必须是以开发出高质量的软件产品为动力的挑剔且善于思考的人，喜欢评估软件产品交付物，并且不受世俗的假设的影响，这个假设就是很多开发人员认为测试人员的工作没有开发工作重要。好的测试人员是善于学习并且渴望学习的人，是具有团队精神的人，是在口头和书面都能够进行有效沟通的人。

12

开发人员的工作成果是实实在在摆在那里的。程序员可以写出代码并且展示给客户，客户一般情况下都会认为程序运行是正确的。从开发人员的角度看，测试的结果只不过是准确性、应用性等方面对质量有些好处。在这种情况下，很多开发人员和测试人员经常是在不太合作的甚至是充满敌意的氛围之中一起工作的。

在很多情况下，开发人员和测试人员的角色都会发生冲突。开发人员的职责是开发成功的程序。测试人员是通过检测缺陷的方式来降低失败的风险，改进软件。开发人员关注技术，因为开发软件的时候要在技术方面花费大量的时间和精力，但是测试人员的动力是为用户提供最好的、能够解决问题的软件。

142

测试人员往往是被忽略的，直到开发周期的末尾，当应用程序即将“完成”的时候才被想起。

测试人员总是对开发的进展感兴趣，并且认识到只有他们从更宽的角度、从多个方面考虑软件质量的时候，质量才能够保证。

12.3.3 项目的目标：把质量保证和开发结合起来

对测试人员来说，把测试活动和开发活动结合在一起的关键是避免给人一种印象：他们是有意把代码搞崩溃或破坏开发工作。理想的情况是，测试人员是衡量软件产品质量的尺子，应该对软件产品进行检查，对其进行评价，考察产品是否满足了客户的需求。他们不是来制造麻烦的，也不是来抱怨产品的，而是来告诉开发人员怎样才能把产品做得更好的。他们给别人的印象应该是“开发人员提高质量的眼睛”。

开发人员需要真正地去关注质量，并且要把测试团队看成开发团队不可或缺的伙伴。开发人员应该认识到，无论他们在开发方面做了多少工作，付出了多少努力，只要软件的质量水平不达标，软件产品就注定要失败。测试经理需要在整个开发周期内不断地提醒项目经理这一点。项目经理需要把这种理念灌输给整个开发队伍。

测试人员必须调整自己的工作与项目进度安排保持一致，与开发工作齐头并进。他们必须要清楚开发工作的具体进展，要参与到所有项目计划和状态会议之中。这样，在开发周期接近尾声的时候，可以降低引进新bug（也称为“副作用”）的风险，也会减少对耗时的回归测试的需求。

必须鼓励测试人员与开发团队中的每个人进行有效的沟通。他们应该与软件的用户建立良好的沟通关系，因为用户可以帮助他们更好地理解软件的质量验收标准。这样，测试人员就能够直接给开发人员提供有价值的反馈。

有空的时候，测试人员应该积极地评审在线帮助和打印出来的手册。让文档作者和测试人员共享文档信息来减轻沟通负担，而不是把这些信息全推给开发人员。

测试人员需要知道软件产品的目标、产品的期望行为和实际的行为、开发的进度安排、任何的建议变更，还有已经报告的问题的状态。

开发人员需要知道什么问题被发现了，软件的哪一部分是可以工作的，哪一部分是不能工作的，用户对软件的感觉如何，将要测试的是什么，测试的进度安排如何，测试资源的可用性怎样，为了测试系统测试人员应该知道些什么，测试工作的当前状态等。

143

质量保证团队与开发团队一起开始工作的时候，测试经理需要与项目经理交谈，并且对齐心协力开发出最好的产品表示出兴趣。下一节描述怎样实现这一点。

12.3.4 迭代/螺旋式开发方法

螺旋式开发方法是对传统的瀑布式系统开发的一种改进，在瀑布式开发方法中，产品开发是通过分阶段按顺序进行的。瀑布式开发模型的一个普遍问题是交付产品可能会延期。

与之相反，螺旋式开发方法则加速了产品的交付。一个具有一定功能的小系统很快就被构建出来并交付给用户，然后在一系列迭代过程中系统得到增强。这种做法的一个优点是用户可以很快看到一些功能，另一个优点是产品在不断的反馈中逐渐成形。例如，用户不需要在开发周期的开始就必须正确地、详细地定义出每个功能，但是需要用户能够对每个迭代进行反馈。

应用螺旋式开发方法，产品会随着时间的推移而持续完善。产品不是静态的，而且从传统意

义上讲可能永远都没有最终的完成。“螺旋”这个术语说明了这样一个事实，传统的按顺序的“分析-设计-编码-测试”等各个阶段会在一个短时间的螺旋或者周期里小范围地实现，然后这些阶段在后续每个周期里重复出现。螺旋式方法经常与原型法和快速应用程序开发相联系。

传统的基于需求的测试总是期望产品定义能够在详细的测试计划之前定稿甚至冻结。在螺旋式开发中，产品定义和规约会无限期地持续完善，也就是说，不会有冻结的规约这种情况。一份内容全面的需求定义和系统设计可能永远也不会完成。

因此，在螺旋式开发环境中唯一可行的测试方法就是“认识到螺旋的本质”。质量保证团队必须与开发团队保持良好的工作关系。测试人员必须密切关注开发人员的工作，一旦有新版本就立刻开始对其进行测试。为了不中断时常发生的产品交付，测试的每次迭代必须是简短的。每次迭代测试的重点必须是先测试增强的和改变的功能。如果螺旋周期的时间允许，自动的回归测试也应该执行，这要求有充足的时间和资源来更新每个螺旋周期的自动回归测试。

客户通常要求对需求变更做出快速的反应，可能还没有正式的发布版本，但也不愿意等待下一个版本来获得新的系统功能。理想的情况是，应该针对产品有一个高效的、自动的回归测试工具，在产品的下一个版本发布之前至少可以做一次快速测试（更多的信息详见第六部分）。

144

螺旋测试是一个从基本的系统开始逐步增量地把系统构建起来的过程。在到达每个阶段末尾的时候开发人员都重新检查整个系统的结构并加以改进。把系统开发的4个主要阶段（计划/分析、设计、编码和测试/交付）画成4个象限（如图12-1所示），描绘了螺旋式方法。相对应的测试阶段是测试计划、测试用例设计、测试开发和测试执行/评估。



图12-1 螺旋测试过程

12

螺旋测试过程从计划和需求分析开始决定系统的功能，然后设计系统基础组件和第一步中确定的功能，接下来，构造系统的功能并对其进行测试。这是螺旋式过程的一次完整的迭代。

完成了第一个螺旋迭代过程之后，用户就有机会检查系统并增强它的功能。这样就开始了螺旋过程中的第二次迭代。这一过程循环地继续下去，直到用户和开发人员都同意系统已经很完善了，过程才进展到实施阶段。

如果系统化地采用螺旋式开发方法，可以有效地保证用户需求能够得到充分处理，也可以保

证用户能够密切地参与到项目中来。系统允许任何业务需求的变更,即使系统开发已经开始。但是螺旋式开发方法有一个主要的缺点:可能永远不能坚定地承诺可以实现出一个完整的运转系统。螺旋总是在一圈一圈地旋转,却永远无法把系统完善到产品级别。这就是经常说到的“死螺旋”。

瀑布式开发的特点是太不灵活,而螺旋式开发的问题恰恰相反。螺旋式方法的灵活性常常导致开发团队忽略了用户真正想要的东西,以至于产品在用户验证的时候失败。这正是质量保证成为螺旋式方法的关键组成部分的原因。质量保证能够确保用户的需求得到满足。

螺旋式方法的一个变种是迭代方法,这种方法强制开发团队必须要在某一个时间点之前实现系统。迭代方法认识到系统不会真正地完成,而是一直在演化。但是,也认识到存在一个系统接近完成、最终用户的需求也基本得到满足的时间点。

在系统开始之前这个实现点就要确定,迭代的次数以及每次迭代过程中定义的目标要做出详细的说明。在最后一次迭代完成的时候,无论处于什么状态系统都必须被实现。

12.4 JAD 的角色

在第一个螺旋中主要交付的是一些目标、初步的功能分解图以及功能规约等。功能规约还包含了外部(用户)的系统设计。众所周知,需求定义的错误和外部设计的错误如果在后期的开发过程中修正会付出巨大的代价。因此,第一次就尽可能获得正确的设计是非常必要的。

有一种称为JAD(联合应用设计)(详见G.19节)的技术可以协助实现这个目标。研究发现JAD与传统的设计技术相比可以提高生产率。在JAD模式下,用户和IT专业人士联合起来在促进会议中一起设计系统。JAD的效果远胜于通过一对一访谈来收集信息的方式。通过将用户置于主导地位,参与者之间的沟通、协调和团队合作等都得到了提高。

JAD从逻辑上可以划分为几个阶段:客户化、会议和综合报告。无论个人在开发中从事什么活动,这些阶段总会存在。每个阶段都有各自的目标。

12.5 原型法的角色

原型法是一种迭代方法,经常用来构建那些用户最初无法准确描述的系统(详细信息参见G.24节)。借助第四代编程语言和应用程序生成器的力量,此概念得以广泛应用。

然而,像其他任何开发方法一样原型法也会有缺点,如果不能系统化地执行,可能会带来更多缺点。与其他任何系统一样,原型必须得到全面测试。如果原型的开发没有建立系统化过程,测试会非常困难。

软件原型多种多样,从简单的输入、处理和输出的文字描述,到完全自动化的版本。软件原型的准确定义目前还没有,这个概念是由几部分组成的。在很多由MIS专业人员定义的特征中有如下几条。

- 构造成本相对便宜(即少于整个系统开发成本的10%)。
- 相对快速的开发可以让原型系统在生命周期的早期得到评估。
- 在实现系统之前为用户提供一个系统关键部分的物理展示。

□ 原型：

- 不会消除或降低对用户需求进行规约和全面分析的需求；
- 不必展现整个系统；
- 仅实现最终产品的功能子集；
- 不必考虑最终系统的速度、界面布置或其他物理特征。

可以说，原型法就是建立系统的实验版本。这些早期的版本可以作为评估有关最终完整系统的想法并做出决策的基础。原型法的一个基本前提就是，在特定的问题领域里（尤其是在线交互系统），应用程序的用户对应用程序应该有些什么功能或应该怎样操作还没有清晰和完整的了解。

通常，在开发过程中没有注意到的错误或缺陷会在系统完全可用后暴露出来。应用程序原型法就是要在开发工作大量投入之前，通过给用户和开发人员提供一种高效沟通想法和需求的方式来解决这些问题。原型法过程使得一组功能性的规约得到充分分析、理解，并被用户、开发人员和管理人员用来决定应用程序是否切实可行以及应该怎样开发。

第四代编程语言使很多机构能够采取原型法技术来实施一些项目。这些语言为原型开发提供了必要的功能，包括定义和管理用户系统接口的用户功能、组织和控制访问的数据管理功能、定义执行控件以及应用程序和物理环境之间接口的系统功能等。

147

最近几年来，原型法的优点逐渐得到认可。原型法有以下优点。

- 原型法强调活动的物理模型，原型无论是外观、感觉还是行为，都像真正的系统。
- 原型法具有非常好的直观性和解释性。
- 维护性能指标、优化访问策略和完善功能这些负担在原型法里都不存在。
- 便于解决有关数据、功能和用户界面的问题。
- 用户通常会比较满意，因为他们最后拿到的是他们见过的东西。
- 很多设计上的注意事项得到重视，设计的高度灵活性很明显。
- 信息需求确认很方便。
- 能够预期变更和错误更正，很多情况下立刻开始实施。
- 需求中不明确和不一致的地方很容易被发现并修正。
- 可以快速消除无用的功能和需求。

12

12.6 开发原型的方法

下面描述一种方法，通过原型的复用以及在开发和使用原型的过程中获得的知识来缩短开发时间。本章不包含螺旋式开发中怎样测试原型，这一内容将在下一节中描述。

12.6.1 步骤 1：开发原型

在螺旋式开发的构造阶段，应用第四代编程语言工具（如Visual Basic或Power Builder）把外部设计和屏幕设计转化成了真实的用户界面。虽然详细的业务功能没有体现在屏幕原型中，但是用户可以通过产生的用户界面的界面感观来想象应用程序是如何工作的。

使用第四代编程语言，开发团队构造了一个原型系统，包含数据录入界面、打印报表、外部

文件例程、特别过程以及过程选择菜单等。所有这些都基于JAD数据建模阶段开发的逻辑数据库结构。使用第四代编程语言完成开发原型系统的任务是一系列迭代发生的事件，如下所述。

定义来自逻辑数据模型的基本数据库结构。按照某些特定的测试要求，数据结构要周期性地载入一些测试数据。

148

定义打印报表的格式。这些格式最初可能由存储在磁盘上的可执行过程文件里的查询命令组成。查询语言的好处是大部分的报告格式可以用第四代编程语言自动完成。原型系统团队只需要为各个报表定义需要打印的数据元素以及选择和排序的标准。

定义交互式数据录入屏幕。这时候是否很好地设计了每一屏幕并不十分重要。以各种格式（如提示、标签、帮助信息，还有输入验证）获取正确的信息是非常重要的。开始时应该尽可能使用默认值。

定义外部文件例程来处理数据，数据可能是批量提交到原型系统的，也可能是原型系统生成而供其他系统处理的。这可以与其他任务并行完成。

定义在原型系统和最终系统中用到的算法和过程。可能包含仅为原型应用而做的一些支持例程序。

定义过程选择菜单。开发人员应该把全部精力都集中在功能上，因为用户会看到这些功能。把表面上看完全不同的过程组合成一个功能是很有必要的，这样用户只用一个命令就可以执行。

定义测试用例以便确定以下各项内容。

- 数据输入验证是正确的。
- 过程和算法可以产生预期的结果。
- 系统的运行被清晰地定义成一系列操作。

不断重复这个过程，增加报告和屏幕格式选项，修正测试中发现的错误，添加对用户的用法说明等。在第二次或第三次迭代之后，或者所做的更改主要是装饰性的而不是功能性的时候，这个过程应该终止。

这时候，原型开发团队应该对系统的全部操作有比较透彻的理解。如果时间允许，团队必须对原型系统的操作和基础结构加以描述。这个很容易通过完成用户手册的草稿来实现。每个界面的拷屏、报表、查询、数据库结构、选择菜单以及分类的过程和算法必须都包含在里面。每个程序的执行用法说明应该包含真正的对话框的插图。

12.6.2 步骤 2：向管理层演示原型

做这个演示的目的是为管理层提供一个决策的机会，管理层对这个系统的战略决策要基于原型系统的外观和目标。演示主要包括对原型系统每个组件及其效果的简短描述以及每个组件典型的用法。如果可能的话，应该为每个观摩演示的人提供一份用户手册的草稿。

149

团队应该强调原型系统的结果以及它对要完成的任务的影响。在这个阶段，系统没有必要是一个能完成最终功能的系统，管理层必须要明白它的局限性。

12.6.3 步骤 3：向用户演示原型

对于是否让最终用户实际使用原型系统，一直有支持和反对两种观点。反对者认为让用户

试用原型系统是有风险的，用户对产品系统的交付的期望值会提高到一个不现实的高度，并且原型系统可能在没有最终完成前就被投入实际生产。一些用户在产品系统最终完成并交付的时候不愿意放弃原型系统，当然，如果原型系统满足了用户的期望并且没有其他争议的话这就不是问题。但支持者认为，用户试用原型系统的时候，能够很快发现程序中的问题和不可接受的系统行为。

应该把原型系统演示给有代表性的用户群体。演示应该包含系统操作的详细描述、结构、数据输入、报表生成和过程的执行。首先，应该让用户明白原型系统不是最终的系统，原型系统是会发生变化的，演示原型系统是为了从用户的角度来发现错误。

演示带来的结果包括变更请求、错误修正以及增强系统功能的一些总体性的建议。一旦演示完成，原型开发团队就会重新开始迭代原型过程中的步骤，变更、修正、增强功能，直到原型系统团队、最终用户和管理层之间达成一致意见。

对于在开发原型系统过程中的每一次迭代，都应该做新的演示以表明系统做了哪些修改，这些修改是来自用户和管理层等各方面的反馈的处理结果。演示会增加用户主人翁的感觉，特别是当他们看到自己的建议在新的原型系统演示中体现出来的时候。因此，对变更的开发和演示应该越快越好。

在演示和使用原型系统的过程中发现的需求，会对系统的范围和目的、系统的概念模型或者逻辑数据模型带来重大的修改。由于这些更改发生在需求规约阶段而不是发生在设计、编码或操作阶段，成本要低得多。

12.6.4 步骤 4：修订并定稿规约

这时候，原型包含了数据录入格式、报表格式、文件格式、逻辑数据库结构、算法和过程、选择菜单、系统操作流，如果可能的话还有一份用户手册。

这一个阶段要交付的内容包括系统需求的正式描述、每个编程对象的第四代编程语言命令文件的清单（即屏幕、报表、数据库结构）、报表样本、数据录入屏幕样本、逻辑的数据库结构、数据词典列表和风险分析等。风险分析包括还没有整合到原型系统中的问题和变更，以及这些问题和变更可能对整个系统的开发和后续操作带来的影响。

原型开发团队评审每个组件以发现不一致、不明确和遗漏的地方，对这些问题加以更正并最终形成正式的规约。

12.6.5 步骤 5：开发产品系统

这时候，开发可以沿着下面的3个方向之一进行下去。

(1) 因为原型系统暴露出了不能克服的问题或系统所要求的环境条件暂不具备，项目被暂停或取消。

(2) 因为不再需要，或者因为效率极低而不能满足生产或维护的要求，原型系统被放弃。

(3) 原型系统开发的迭代过程继续进行下去，在每一次迭代中增加系统的功能和优化系统的性能，直到原型系统演化成产品系统。

如何进行下去的决策通常基于下列因素做出。

- 原型系统的实际成本。
- 在原型系统开发过程中暴露出的问题。
- 维护资源是否到位。
- 组织中的软件技术是否到位。
- 行政方面的和组织方面的压力。
- 对原型系统的满意度。
- 把原型系统转变成产品系统的难度。
- 硬件需求。

12.7 持续改进螺旋测试方法

软件测试的目标是找出实际状况和期望状况之间的差异，也就是去探寻软件中的缺陷。通过测试能够找出那些还没得到满足的用户需求和因缺陷而受到影响的功能。人们普遍认为测试目标是发现bug，但这是对测试目标的一个有局限的定义。测试不仅仅是要发现bug，还必须把bug放入一个框架中，使测试人员能够预测软件的工作方式。

在螺旋式和快速应用开发的测试环境中，可能没有最终的系统功能需求，这些需求是非正式的并且是逐步完善的。同样，测试计划也可能直到产品发布才最后完成。想拥有一段相对较长的时间，依照完整的需求规约来创建测试计划是不可能的。测试是一个持续改进的过程，随着系统的变更而频繁地发生。产品随着时间的变化而不断地日趋完善，而不是静态的。

测试组织需要深入了解开发工作，并且与开发人员密切配合，一起工作。一旦有新版本可用就马上对其进行测试。首先要测试新增强功能，或为解决上一个螺旋中报告的缺陷而修改过的软件。如果时间允许，还应该执行回归测试，以保证系统中其他部分没有版本回退。

在螺旋式开发环境中，软件测试再次被描述成一个持续改进的过程，这个过程必须集成到快速应用开发方法中。把测试看做是开发的集成功能，以防止开发在没有测试的情况下往前进行。应用PDCA模型的Deming持续改进过程原理将再次被应用到软件测试过程中（见图12-2）。



图12-2 螺旋测试与持续改进

在持续改进过程开始之前，测试部门需要执行一系列的信息收集计划步骤来理解开发项目的目标、当前状态、项目计划、功能规约以及风险等。

一旦这些事情都做完了，持续改进过程正式的“计划”步骤就开始了。这一步的主要工作是撰写软件测试计划。测试计划是做好测试的基础，应该被看成是一个不断完善的文档，也就是说，随着系统的变更，测试计划也要完善。一份好的测试计划的大纲应该包括引言、整体计划、测试

需求、测试规程以及测试计划中其他的细节。这些内容还可以进一步地细化成业务功能、测试场景和脚本、功能/测试矩阵、期望的结果、测试用例检查表、问题报告、需要的软件、硬件、数据和人员、测试进度表、测试的进入标准、退出标准以及总结报告等。

测试计划的内容定义得越清晰，“计划”步骤越容易完成。如果在测试计划完成之后，具体的测试工作开展之前，系统发生了变化，测试计划也应该相应地进行更新。

持续改进过程的“执行”步骤包含测试用例设计、测试开发和测试执行。这一步中描述了怎样设计测试计划中的测试用例和怎样执行测试活动。设计包括功能测试、GUI测试，阶段性系统和验收测试。一旦整体的测试设计完成了，就开始进行测试开发，包括创建测试脚本和程序来为测试用例提供详细内容。

测试团队负责执行测试活动，并且必须保证这些测试活动是按照测试设计执行的。“执行”步骤也包括测试的设置、对新旧测试的回归测试以及记录所有发现的缺陷等。

持续改进过程的“检查”步骤包括度量的准则和分析。正如第5章中所描述的，对Deming方法来说至关重要的是需要做一些基本决策，这些基本决策应该尽可能多地基于准确、及时的数据。度量是验证工作量和测试进度是否按时的关键，也是发现任何新的资源需求的关键。

在“检查”这一步骤中，发布一些中间的测试报告是很重要的。这包括记录测试结果，并把测试结果与测试计划和测试目标相关联。

持续改进过程的“改进”步骤包括对下一次螺旋迭代过程的准备。在这一步骤里会改进功能/GUI测试、测试套件、测试用例、测试脚本和阶段性系统测试和验收测试，如果需要的话，也会改进缺陷跟踪系统和版本控制系统。这一步骤还包括设计一些合适的度量方法来检查没有按照计划执行的工作或不是原来预期的结果，例如对测试团队、测试规程和测试的技术范围的重新评价等。所有这些都是对更新过的测试计划的反馈。

经过几次测试螺旋过程的验证，应用程序的功能已经稳定，这时全面的系统和验收测试就要开始了。这些测试通常是可选的。系统测试计划和验收测试计划分别开发以定义测试目标和将要完成的特定测试。

持续改进过程的最后活动是总结和报告螺旋测试的结果。在所有测试的末尾都应该有一个主要的测试报告。无论是中期报告还是最终报告，撰写的过程都是一样的。另外，与测试中的其他任务一样，报告撰写也应该注意质量控制。但是，最终的测试报告应该比中间的测试报告全面得多。对于每种测试，这个报告都应该描述发现的缺陷的记录、数据精简技术、根本原因分析、结果开发以及对于当前和未来项目的后续建议。

图12-3通过把每一步与PDCA质量模型相关联的方式提供了对螺旋测试方法的总体描述。附录A提供了各部分的详细叙述。螺旋测试方法提供了在这种环境下的一个测试框架。主要步骤包括信息收集、测试计划、测试用例设计、测试开发、测试执行/评价、准备下一轮螺旋。该方法包括一系列与每一步相关联的任务或检查表，测试组织可以根据其需求进行选择。螺旋测试方法对系统功能进行彻底的测试。螺旋测试完成以后，接下来是典型的系统测试、验收测试和总结报告。

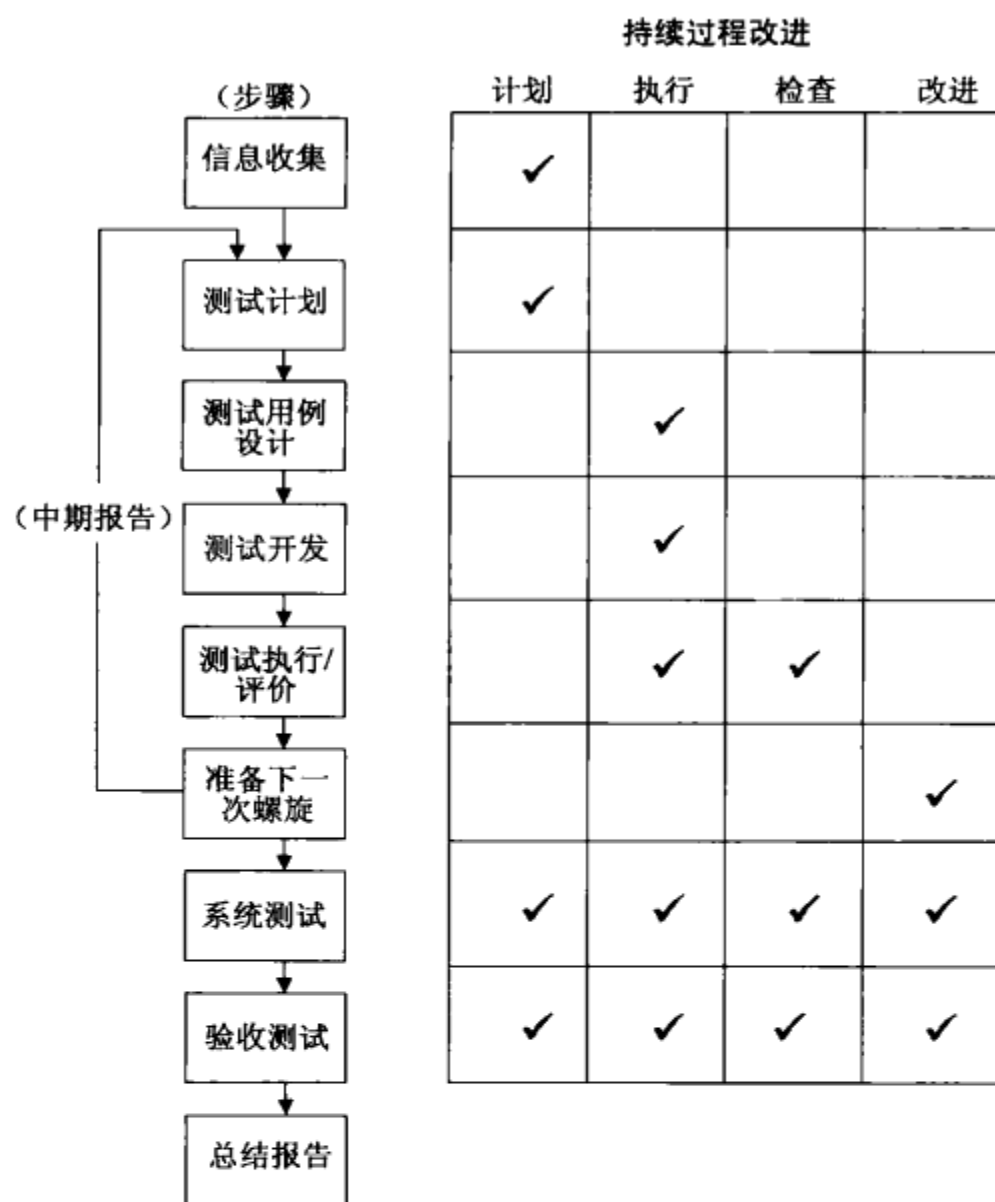


图12-3 螺旋测试方法

回忆一下，在螺旋式开发环境中，软件测试可以描述为必须集成到快速应用开发方法中的持续改进过程。应用PDCA模型（见图13-1）的Deming持续改进过程也将应用到软件测试过程中，我们现在探讨螺旋式模型中的“计划”部分。

图13-2列出了与螺旋测试中“计划”部分中的信息收集相关联的步骤和任务。在描述每一个步骤和每一项任务的同时，也给出了一些有价值的提示和技术。



图13-1 螺旋测试与持续改进

信息收集的目的在于理解软件项目开发的范围，获取开发项目的相关信息以及对这些信息进行组织，然后开始撰写测试计划。在项目开发的过程中，如果需要的话也可以进行其他的访谈。

适当的准备工作对访谈的成功是很关键的。在访谈之前，清楚地确定访谈目标并与各方沟通好这些目标、确定主导访谈和负责记录的质量保障人员、计划好时间和地点、准备好所需要的介绍材料、与开发方面预先沟通好所需要的东西等，都是非常重要的。

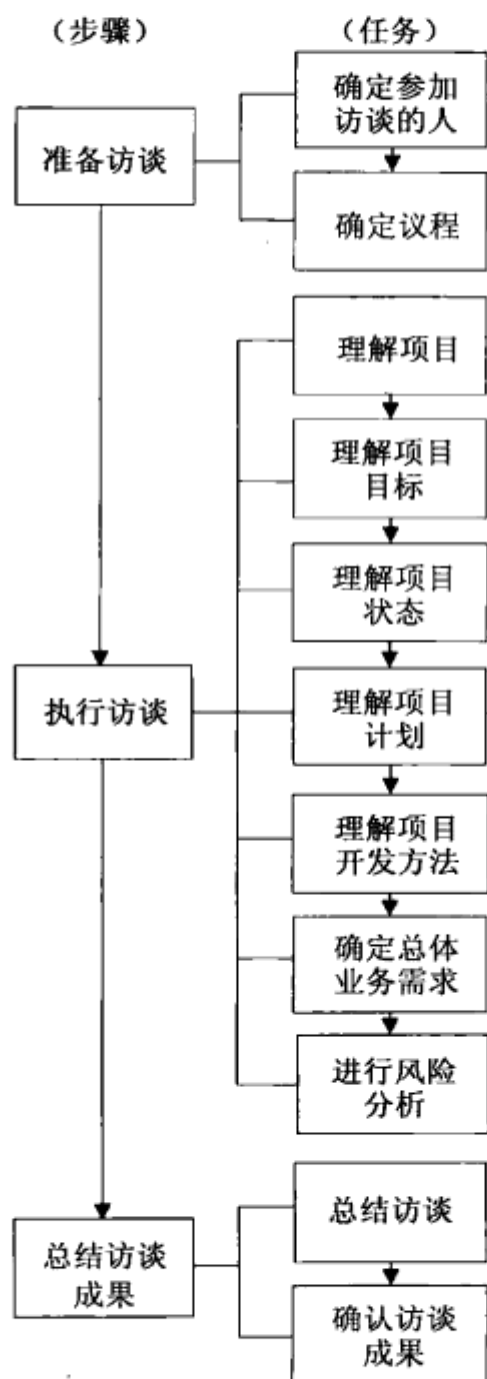


图13-2 信息收集（步骤/任务）

尽管很多访谈都不是结构化的，但图13-2所示的访谈步骤和任务会很有帮助。

13.1 步骤 1：准备访谈

13.1.1 任务 1：确定参加访谈的人

建议质量保证方面的访谈人员不要多于两个，其中一个人的作用是负责提问，而另一个人则负责做详细的记录。这样分工的好处是能够让提问者集中精力挖掘信息。理想的情况下，提问的人应该是负责项目测试工作的经理，负责记录的人应该是分配到这个项目中的测试工程师或测试主管，配合提问的人进行访谈，并且记录所有相关的信息以及问题、假设和疑问的详细列表。

建议开发方面的参加人员包括项目负责人、开发经理或其他资深的团队开发成员。虽然开发团队的成员可以做笔记或记录，但这是专门负责记录的人的责任。如果不止一个人去做记录的话，结果会导致一些混乱，因为多份记录最终也要整理成一份。对负责记录的人来说，最有效的方法是不但要做记录，而且还要在访谈结束的时候做个总结。（参见F.20节，项目信息收集检查表可以用来验证信息的有效性，是项目开始阶段所要求的。）

13.1.2 任务 2：确定议程

成功访谈的一个关键因素是有一个经过深思熟虑的会议议程表。议程表应该是由访谈者提前准备好，并由开发负责人同意。议程表应该包括引言、应该覆盖的关键点和总结等部分。议程表的主要目的是让测试经理能够收集足够的信息来仔细研究质量保证的活动，并且可以开始编写测试计划。表13-1给出了一个议程表的例子。（详细信息在步骤2中描述。）

表13-1 访谈议程表

I.	引言
II.	项目概述
III.	项目目标
IV.	项目状态
V.	项目计划
VI.	开发方法
VII.	总体需求
VIII.	项目风险和问题
IX.	总结

13.2 步骤 2：执行访谈

成功的访谈包含几个特定的元素。第一个元素是定义要讨论什么，即“谈论我们将要谈论的内容”；第二个元素是讨论具体的详细内容，即“谈论具体内容”；第三个元素是总结，即“谈论

我们曾经谈论过的内容”；最后一个元素是时间安排。访谈者应该声明事先估计的访谈持续时间，如果时间到了但是没有完成议程表上所安排的全部项目，应该再安排一次后续的访谈，这是一条基本原则。当访谈进展到了解详细信息的时候，很难做到这一点，但还是应该遵守这条原则。

13.2.1 任务 1: 理解项目

在了解项目详细信息以前，访谈者应该阐明访谈的目标，并且把议程安排表达清楚。与其他任何类型的访谈一样，访谈者应该事先声明同一时间内应该只有一个人发言，别人不要随便打断，除非得到发言者的许可。还应该声明的是，大家应该只关注发言的内容。

然后访谈者要向大家介绍自己和负责记录的人，并让开发团队的成员作自我介绍。每个人要介绍自己的姓名、职务、角色和工作职责，还有对访谈的期望。访谈者应该指出本次任务的目的是获得基本的项目背景信息。

应该提问下列通用的问题以获得基本的信息。

- ☐ 项目的名称是什么？
- ☐ 项目的总体目标是什么？
- ☐ 项目的用户是谁？
- ☐ 项目是什么时候开始的？
- ☐ 项目的预期完成时间？
- ☐ 项目在什么地点开发？
- ☐ 项目的工作量有多少个人月？
- ☐ 这是一个新项目、维护性的项目还是打包开发的项目？
- ☐ 主要的问题、议题和关注点是什么？
- ☐ 是否有计划来解决问题和议题？
- ☐ 项目的预算是否按时？
- ☐ 项目的预算太紧张、太宽松还是正好？
- ☐ 组织内哪些单位要参加项目？
- ☐ 绘制组织结构图了吗？
- ☐ 给每个单位分配哪些资源？
- ☐ 决策的结构是什么？也就是说，谁负责决策？
- ☐ 项目中有哪些角色？每一个角色相关的职责是什么？
- ☐ 测试团队在日常工作中将要与谁交流？
- ☐ 是否已做了质量管理计划？
- ☐ 是否建立了定期的评审规程？
- ☐ 是否有来自用户群体的代表来描述质量？

13.2.2 任务 2: 理解项目目标

要制定一个开发项目的测试计划，理解项目的目标是很重要的。本项任务的目的是理解项目的范围、要求和总体需求。

应该提问下列问题以获得基本的信息。

- 目的。
 - 正在开发的系统是什么类型的？例如，工资系统、订单录入、库存，还是应收账款/应付账款？
 - 为什么要开发这个系统？
 - 包含什么子系统？
 - 主观需求是什么？例如，易用性、效率、士气、灵活性？
- 范围。
 - 谁是系统的用户？
 - 用户的工作职务和角色是什么？
 - 系统的主要功能和子功能是什么？
 - 哪些功能不用实现？
 - 在系统的范围之内要处理哪些业务流程？
 - 有分析图表（如业务流图、数据流图或数据模型等）来描述系统吗？
 - 项目交付物已经和完成标准一起定义好了吗？
- 收益。
 - 系统带给用户的预期收益是什么？
 - 提高生产率；
 - 改进质量；
 - 节省费用；
 - 增加收入；
 - 更多的竞争优势。
- 战略。
 - 战略或竞争优势是什么？
 - 系统会对组织、客户、法律、政府等有什么影响？
- 约束。
 - 系统在财务、组织、人力资源、技术等方面有什么约束或限制？
 - 哪些业务功能和流程是在系统的范围之外的？

13.2.3 任务 3：理解项目状态

本项任务的目的是理解项目目前处于什么状态，这将有助于规划测试工作。例如，如果这是第一次访谈，并且项目已经处于编码阶段了，那么测试工作的进度就已经落后了。

应该提问下列问题以获得基本的信息。

- 制定详细的工作计划了吗？包括活动、任务、依赖、资源分配、工作量的估计、里程碑等。
- 项目的进度如何？
- 完成项目的时间很紧张吗？

- ☐ 完成项目的时间很宽裕吗?
- ☐ 完成项目的时间正好吗?
- ☐ 存在影响关键路径的重要的进度延迟吗?
- ☐ 项目离达到目标还有多远?
- ☐ 用户要求的功能和对质量的期望现实吗? 能够满足吗?
- ☐ 项目工作量的完成情况与进度安排相符吗?
- ☐ 项目的费用在预算范围之内吗?
- ☐ 哪些开发交付物已经交付了?

13.2.4 任务 4: 理解项目计划

因为测试工作要跟踪开发的情况, 所以理解项目的工作计划对测试来说是很重要的。应该提问下列问题以获得基本的信息。

160

- ☐ 工作分解。
 - 有一个用 Microsoft Project (或其他工具) 做出的项目计划吗?
 - 计划详细到什么程度? 例如, 确定了多少主要的任务和子任务?
 - 项目有哪些主要的里程碑 (包括内部的和外部的)?
- ☐ 资源分配。
 - 是否已经为每项工作都分配了合适的资源?
 - 工作计划做好平衡了吗?
 - 对阶段性资源做计划了吗?
- ☐ 进度。
 - 项目是按计划的进度进行吗?
 - 项目的进度与原计划相比落后了吗?
 - 项目的计划有周期性的更新吗?

13.2.5 任务 5: 理解项目开发方法

测试工作必须与开发方法相结合。如果把测试看做是一个独立的职能部门, 可能得不到相应的资源和重视。把测试看做是综合职能部门, 可以有效地防止没有测试就进行开发。通过对总体任务的增加或修改, 可以把测试的步骤和任务整合到系统开发方法之中。特别要注意的是, 测试部门应该知道在系统开发方法中测试设计什么时候开始, 也应该知道什么时候系统达到能运行的状态, 并且可以开始记录并修正发现的缺陷。

应该提问下列问题以获得基本的信息。

- ☐ 采用的方法是什么?
- ☐ 开发团队采用的是什么样的开发方法和项目管理方法?
- ☐ 对于开发方法, 开发团队执行情况如何?
- ☐ 有采用变通方法的余地吗?
- ☐ 关于标准。

- 标准和实际的做法都有记录文档吗？
- 标准有用还是妨碍了生产效率？
- 开发团队执行标准的情况怎样？

13.2.6 任务6：确定总体业务需求

161 软件需求规约详细地定义了软件产品在特定环境里的功能。在不同的开发组织里，需求规约的形式也不尽相同，有的只是泛泛地定义了应用程序将要实现的功能，有的则对功能进行详细地描述，请参见附录C。无论是哪一种情况，测试经理都必须评估开发项目的范围以便开始制定测试计划。

应该提问下列问题以获得基本的信息。

- 总体功能是什么？总体功能应该一一列举出来。例如，订单处理、财务处理、报表功能、财务计划、采购、库存控制、销售管理、运输、现金流分析、工资、费用以及招聘系统等。这个清单定义了应该实现什么功能，同时提供给测试经理一个概念，那就是测试设计和实现要达到一个什么样的水平。访谈者应该尽可能地获得详细的信息，包括每一个功能的详细分解。如果在访谈中得到的信息不详细，应该请求对功能进行详细分解，必须指出的是，这些信息对准备测试计划而言是很重要的。
- 对系统有什么要求（或最低要求）？应该提供对操作系统（如Windows）版本的要求，以及对CPU、磁盘空间、内存以及通信硬件的最低要求。
- 用到哪些Windows接口或外部接口？规约应该从外部视角定义应用程序的行为，通常是通过定义输入和输出来实现的。规约还应该包括与其他应用程序或子系统的接口的描述。
- 性能要求是什么？包括对速度、稳定性、数据量吞吐率、响应时间、各种功能的恢复时间以及压力等方面的描述。这些内容是理解性能测试和压力测试要做到什么程度的基础。
- 有其他质量特性需要进行测试的吗？包括可移植性、可维护性、安全性以及易用性。这些内容是理解其他系统级别测试要做到什么程度的基础。
- 有设计方面的约束吗？包括对操作环境、数据库完整性、资源限制、实现语言标准等方面的限制的描述。

13.2.7 任务7：进行风险分析

162 这个任务的目的是衡量一个应用程序系统的业务风险程度，以改进测试。有两种实现方法：识别出高风险的应用程序，进行更高强度的测试，或者帮助识别出容易出错的应用程序模块，重点测试这些模块。该任务描述了如何使用风险分析技术来对当前测试的应用程序进行风险分析。

1. 计算机风险分析

风险分析是用来识别危险区域（即可能造成潜在损失的区域）的正式方法。危险区域指的任何可能会被故意或者偶然误用而对公司造成损失的弱点。风险的识别促使测试过程衡量这些危险区域的潜在影响。（例如，风险或者危险区域可能造成的最大损失有多少。）

风险永远是测试工作需要考虑的部分。通常我们习惯于先预测可能出现的问题，然后对这些

问题进行测试以决定是否需要额外的资源和关注。然而，风险分析方法经常是不正式而且效率低下的。

通过适当的分析，测试经理应当能够预计出发生不利结果的可能性，比如：

- 预期的收益全部或者部分无法获得；
- 成本和进度超出预计；
- 组织内部控制不完善；
- 最终系统的性能明显低于预期；
- 系统与所选的硬件、软件不兼容。

下面回顾一下风险分析的不同方法和计算机风险的维度，然后描述指定风险优先级的方法。有如下3种进行风险分析的方法：

2. 方法1：判断和本能

这个方法用来决定需要执行多少测试，测试人员通过与过去项目的比较来评估当前项目的风险程度。虽然这种方法很有效，但它所需要的知识和经验是无法传承的，只能慢慢培养。

3. 方法2：经济评估

风险是造成损失的可能性。这种可能性可以通过下面的公式表示：

$$\text{发生的频率} \times \text{每次发生造成的损失} = \text{每年损失的期望数量}$$

这个公式可以把业务风险量化为经济损失。然而，一般我们都不使用公式而是通过感觉来估计发生问题可能会造成的经济损失。用经济损失来衡量项目风险的缺点在于确切数目（即发生的频率和每次发生造成的损失）一般都是很难估计的，而且这种方法也意味着需要很高的精确度，但实际上很难达到。

4. 方法3：确定并评估风险属性

经验表明造成潜在风险的主要因素是项目规模、使用相关技术的经验和项目结构。项目的投资、员工数量、用时以及影响的部门越多，风险就越大。

由于无法预期的技术问题出现的可能性很大，项目风险随着项目团队对于硬件、操作系统、数据库和应用程序语言的熟悉程度的降低而增加。同样一个项目，对于一个大规模的、技术领先的系统开发部门来说是低风险的，但是对于一个小规模的、技术相对落后的团队来说则可能是高风险的。不过，对于项目中涉及的有着广泛的商业应用的技术，后者可以通过购买外部技能来降低风险。

在一个高度结构化的项目中，任务的性质在一开始就完全决定了项目的输出。在整个项目的生命周期中，我们都在不断地工作以完成这个输出。这类项目的风险就比那些输出更多地受经理的判断和项目变更的影响的项目要小得多。

这些因素之间的关系用权重来表示，测试经理可以使用这种加权的总分所代表的风险程度来对应用程序进行排序。例如，这种方法可以显示出应用程序A比应用程序B的风险高。

风险评估，首先要评估单个风险属性的权重。例如，如果一个因素的重要性是另一个的两倍，那么它的权重就要乘以二。计算的结果用来和同一开发部门的其他项目的得分进行比较，以决定应用程序之间的相对风险，但是并不能用来测定绝对风险。

表13-2使用对风险属性加权的方法比较了3个项目。项目规模的权重为2，使用相关技术的经验的权重为3，项目结构的权重为1。每个项目的得分分别和相应因素的权重相乘，可以明显看到项目A的风险最高。

表13-2 确定风险属性并为风险属性加权

因素 权重	项目A（得分×权重）	项目B（得分×权重）	项目C（得分×权重）
项目规模（2）	$5 \times 2 = 10$	$3 \times 2 = 6$	$2 \times 2 = 4$
使用相关技术的经验（3）	$7 \times 3 = 21$	$1 \times 3 = 3$	$5 \times 3 = 15$
项目结构（1）	$4 \times 1 = 4$	$6 \times 1 = 6$	$3 \times 1 = 3$
总分	35	15	22

风险分析阶段收集的信息可以用来分配测试应用程序系统所需的测试资源。举例来说，高风险的应用程序需要高强度的测试，中等风险的系统需要少一些的测试，低风险的系统只需要最少的测试。应当基于高风险的因素来选择测试区域。举例来说，如果计算机技术是一个高风险的因素，测试经理可能就希望花更多的时间来测试开发团队使用该项技术的效率。

13.3 步骤 3：总结访谈成果

13.3.1 任务 1：总结访谈

在访谈结束以后，访谈者应该回顾一下访谈的议程并大概列出主要的结论。如果需要进一步安排下一次访谈，访谈者应该趁所有成员都在场约好下次访谈的时间。

通常的情况是，在访谈的过程中所做的记录结构上比较混乱，除了记录者本人，其他人很难去整理清楚。但是，记录的大概顺序必须是和议程相吻合的。在访谈结束的时候，应该把记录形成一份正式的总结报告，这项工作应该由记录人来完成。目的是把这一阶段的结果尽可能搞清楚，以便于质量保证和开发团队使用。要注意的是，负责整个访谈工作的领导必须要对这些材料加以评审、完善和修改。（参见E.16节，会议纪要适用于在项目信息收集阶段把访谈结果和下一步的工作内容文档化。）

13.3.2 任务 2：确认访谈成果

本项任务的目的是让访谈者和开发团队之间达成一致从而保证对项目有一致的理解。在访谈记录被整理成正式报告以后，把总结报告分发给所有参加访谈的其他成员是很重要的，诚挚地征求他们的意见和建议。访谈者要积极地对待意见一致和不一致的地方，对报告任何修改都要得以落实。一旦大家都达成了一致，访谈者应该提供出总结报告的副本给大家。

项目测试计划的目的是，在一种有组织的方式下为完成测试打好基础。从管理的角度来看，测试计划是最重要的一份文档，因为它帮助管理整个项目。如果全面而谨慎地设计一份测试计划，那么测试的执行和结果分析过程将会非常顺利。（参见E.1节，有一个单元测试计划的例子；参见E.4节，有一个系统测试计划的例子；参见F.24节，有一个用来检验单元测试是否彻底和全面的单元测试检查表。）

测试计划是一个不断更新的文档，尤其是在螺旋式环境中，因为系统在不断地变化。系统改变时，测试计划也要随之改变。一份好的测试计划应该具有下列特点。

- 有机会发现大部分缺陷。
- 为绝大部分代码提供测试覆盖。
- 具有灵活性。
- 可以很容易且自动地执行，并且具有可重复性。
- 定义要执行的测试种类。
- 清晰地记录期望的测试结果。
- 当发现缺陷时提供缺陷修改时间。
- 清楚地定义测试目标。
- 阐明测试策略。
- 清楚地定义测试退出标准。
- 没有冗余。
- 识别出风险。
- 记录测试需求。
- 定义测试交付物。

有很多编写测试计划的方法，图14-1展示了一个框架，其中包括了基本的测试计划中要考虑的绝大部分事项。可以把它当做测试相关项的一个检查表。这些项中的部分内容明显是必需的，例如定义测试需求和测试团队，而有些条目可能就不一定是必需的，这取决于项目的特性以及时间约束。

规划测试方法包含以下3个步骤：建立项目测试计划，定义度量标准，评审和批准项目测试计划。这3个步骤可以分解为各自相关的一系列任务，如图14-1所示。

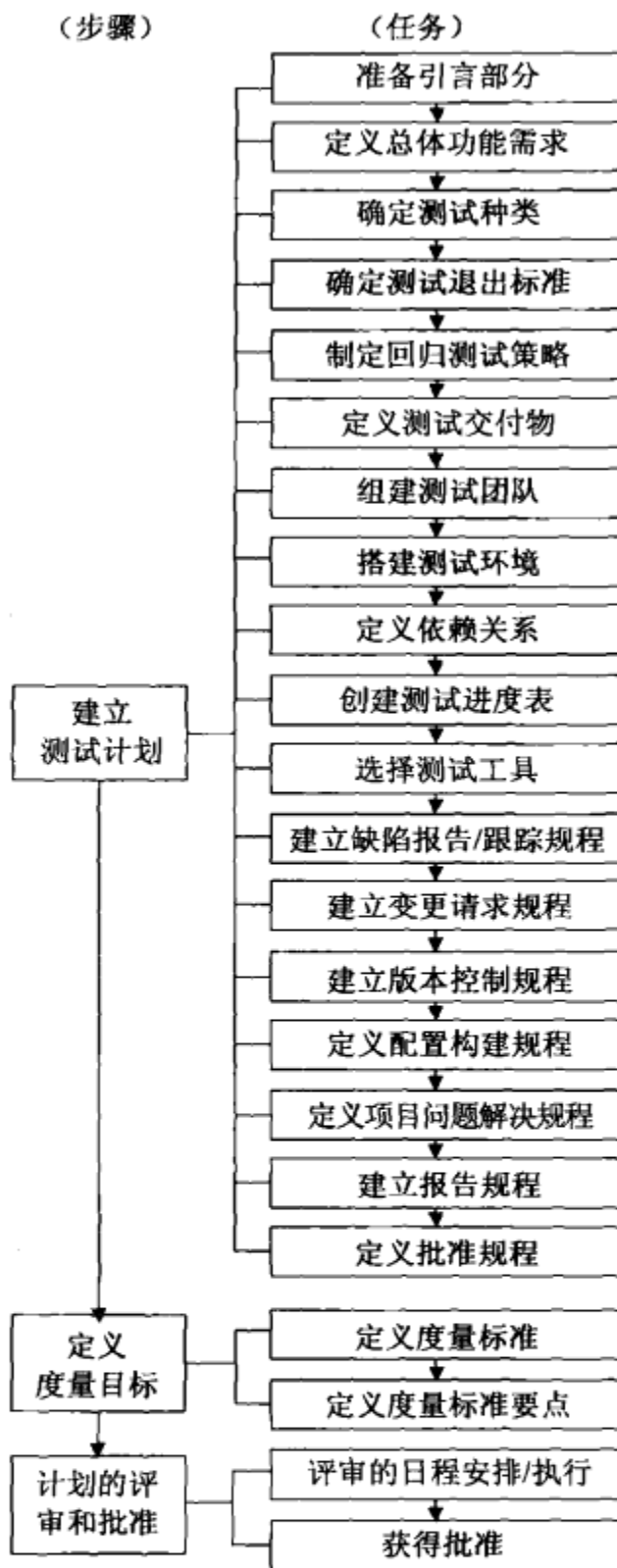


图14-1 测试计划（步骤/任务）

14.1 步骤 1：建立测试计划

14.1.1 任务 1：准备引言部分

测试计划详细内容的第一部分是对将要在相关机会应用中解决的问题的描述。这部分定义了项目的简要背景，描述了促成系统开发决策的事件或当前状态。同时，在引言部分中还应给出应

用程序开发的风险、用途、目标和收益，以及整个团队的评价成功的因素。关键成功因素是一个可测量项，对于评价一个主要功能是否达到预期目标有着决定性的影响。目标是整个团队经过努力达到的一个可测量的最终状态。目标的例子包括下面一些。

- 新的产品机会。
- 效率的提高（内部的和外部的）。
- 企业的形象。
- 成长（内部的和外部的）。
- 经济方面（收入、成本收益率等）。
- 竞争地置。
- 市场领先。

引言部分中同样需要包含关于管理的概要描述。项目发起人是整个项目拥有最终决定权的一个角色。这个角色在如下方面对于项目拥有授权：项目资金、项目成果、解决项目冲突问题等，还要对整个项目的成功负责。管理概要描述了从管理者角度看到的适当的应用程序。应当描述将要解决的问题、应用程序的目标、商业机会等。在目标中应当指出该应用程序是否是对旧系统的替代，同时该应用程序可能在管理、技术等方面对整个组织产生影响，如果有影响也要记录下来。

应当列出所有可能用到的文档并且描述出其状态。例如，需求规约、功能规约、项目计划、设计规约、原型、用户手册、业务模型/流图、数据模型、项目风险评估等。项目风险是对项目开发的潜在不利影响，这里还应列出跟测试活动相关的项目风险。例如，测试技能的缺乏、测试工作的范围、自动化测试工具的缺乏以及其他一些类似的风险。更多详细内容参见E.4节。

168
169

14.1.2 任务 2: 定义总体功能需求

功能规约由以下各部分组成：功能的层次分解、功能窗口结构、窗口标准以及待开发系统的最低运行需求。窗口标准的一个例子是Windows 95的图形用户界面标准。最低运行需求的例子可能是：Windows 95、奔腾二代的微处理器、24 MB内存、3 GB硬盘空间、一个调制解调器。开发工作进行到这里，一个完整的功能规约可能还没有被定义出来，但需要关于基本窗口结构的最少业务功能的一个清单。

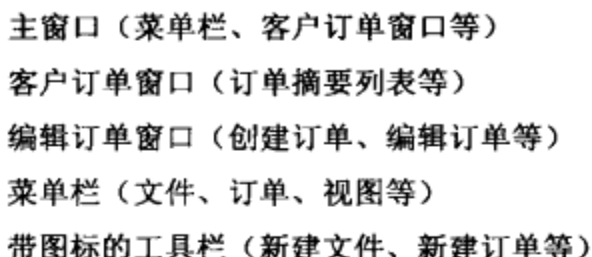
一个基本的功能清单包含系统的主要功能，每个功能都要以“动宾”结构命名和描述。这个清单是构建功能测试的基础（见图14-2）。

订单处理（比如创建新订单、编辑订单等）
客户处理（比如创建新客户、编辑客户等）
财务处理（比如接受付款、存储付款等）
库存处理（比如获得产品、调整产品价格等）
报表（比如创建订单报表、创建账目接受报表等）

图14-2 总体业务功能

功能窗口结构描述了在窗口环境中如何实现功能。这个时候，完整的功能窗口结构也许还没

有，但是一个主要窗口的列表应该已经有了（见图14-3）。



主窗口（菜单栏、客户订单窗口等）
客户订单窗口（订单摘要列表等）
编辑订单窗口（创建订单、编辑订单等）
菜单栏（文件、订单、视图等）
带图标的工具栏（新建文件、新建订单等）

图14-3 功能窗口结构

14.1.3 任务 3：确定手动/自动化测试的类型

需要设计和执行的测试种类完全取决于应用程序的目标，也就是整个团队一直努力达到的可测量的最终状态。举例来说，如果是一个金融方面的应用程序，有大量的用户将要使用这个系统，那么我们就需要进行特别的安全性和易用性测试。无论如何，有3种类型的测试是几乎任何应用程序中都需要的：功能测试、用户界面测试和回归测试。功能测试占测试工作量的一大半，主要关心功能是否能够正确地执行。这是一个基于黑盒模型的测试，测试者完全不关心应用程序内部的行为和结构。用户界面测试，或者称为图形用户界面测试，检查用户交互功能或者功能窗口结构。它保证对象状态与功能有适当的依赖关系，并且在各功能之间提供了有用的导航。回归测试根据在调试、维护或者新版本发布时所做的变更对应用程序进行测试。

其他需要考虑的测试类型包括系统测试和验收测试。系统测试是对整个系统功能性、性能和整体适用性的最高级别的测试和评估。验收测试是一个由用户执行的可选测试，目的在于证明应用程序的能力满足了用户的需求。这个测试是否执行取决于项目的形式。有时系统测试就足够了。

最后，需要定义能够使用测试工具自动进行的测试。自动化测试具有3点好处：可重复性、杠杆作用、功能可扩展性。可重复性保证了测试可以反复执行多次，每次执行都保持一致。杠杆作用来自于先前执行的那些测试的可重复性，以及可以用工具编排测试的执行顺序，这是自动化测试的独有特征。随着应用程序的不断更新，越来越多的功能被添加进来。使用自动化测试，功能的覆盖通过测试库来维护。

14.1.4 任务 4：确定测试退出标准

最困难的也最具有政策性的问题就是决定何时停止测试，因为要知道何时能够发现所有的缺陷是不可能的。至少有如下4个退出测试的标准。

(1) 计划的测试时间期满——这个标准是很弱的，因为它与验证应用程序的质量无关。这个标准根本没有考虑也许测试用例不够充分或者也许不再有可以轻易发现的缺陷了。

(2) 发现了预先确定的缺陷数量——这个标准的问题在于如何估计能够发现的缺陷数量以及可能会过高地估计缺陷数量。如果缺陷的数量估计得不足，那么测试就会不完全。可能的解决方案包括凭借同一支开发团队开发相似应用程序的经验、预测模型以及行业的平均情况等。如果缺陷的数量估计得过高，那么测试工作也许在合理的时间范围内永远无法完成。一个可能的解决方

法是估计完成时间，在图上标示出单位时间发现的缺陷数。如果发现缺陷的速度明显减慢了，这也许就是一个“完结”点，表明绝大部分的缺陷已经被发现了。

(3) 执行所有的正常测试都不再发现任何缺陷——这个标准的主要问题是，在这种方式下，测试人员不会很主动地去设计破坏性的测试用例，以促使被测程序达到它的设计极限。举例来说，当测试程序不再发现缺陷的时候，测试人员的工作就完成了。测试人员期望不要发现缺陷，而且可能下意识地设计那些能够表明程序已经没有缺陷的测试用例。当我们有一组严格的、非常全面的、接近百分之百覆盖的测试用例时，这个标准是有效的。这个标准的问题就是确定何时我们拥有一套完整的测试用例。如果觉得正是这种情况，这时一个好的策略是继续进行随机测试。随机测试通常是黑盒测试技术，测试人员完全放开自己的思维，在头脑中列举尽可能多的测试场景。经验表明这项技术是非常有用的附加性技术。

(4) 以上3项的结合——大部分测试项目都利用了以上3个退出标准的结合。我们建议所有的测试都应该被执行，但是随机测试应该根据项目的进度情况来安排。

14.1.5 任务 5: 制定回归测试策略

回归测试是根据螺旋式开发过程、调试、维护或者新版本发布时所做的变更对应用程序进行的测试。这种测试应该在系统的功能更新或修复已经完成之后进行，以保证所做的变更没有产生预计之外的影响。比如，对如下错误的更正必须要进行回归测试：逻辑和控制流的错误、计算错误、接口错误等。装饰物（比如图标）错误一般来讲不会影响其他的功能因而不需要回归测试。

对于每次螺旋过程，测试集合中的所有测试都重新执行一遍是最理想的，但是由于时间的限制，一般来讲是不现实的。在螺旋式开发过程中，比较好的回归测试策略是每次螺旋过程都执行一些回归测试，这些测试要确保后面的螺旋开发或者错误更正没有影响先前演示的功能。在系统已经稳定和所有功能已经被确认之后的系统测试里，回归测试应该由系统测试的一个子集组成。我们需要创建一套策略来决定应该包含哪些测试用例（参见E.21节）。

再测试矩阵是一个非常好的工具，如表14-1所示，它将测试用例和功能（程序单元）联系起来。矩阵中的每个检查标记（对勾）都表明当对应的功能（或程序单元）因为增强或者修改时，标记的对应测试用例应当被重新测试。可以在第一次测试螺旋之前建立再测试矩阵，但是要在之后的螺旋过程中维护好这个矩阵。随着在一个螺旋式开发过程中功能（或程序单元）的改动，应当检查和创建再测试矩阵中现有的和新的测试用例，为下一次螺旋做好准备。在后续的螺旋过程中，一些功能（或程序单元）也许很稳定，在一段时间内不会再更改。在测试螺旋过程之间，应当考虑有选择性地去掉一些相关的检查标记。另外参见E.14节。

表14-1 再测试矩阵

	测试用例				
	1	2	3	4	5
业务功能					
订单处理					
创建新订单	√	√	√	√	

(续)

	测试用例				
	1	2	3	4	5
完成订单					
编辑订单	√			√	
删除订单					
客户处理					
创建新客户					
编辑客户					
删除客户		√			
财务处理					
接收客户付款		√	√		√
存储付款					
支付供应商					
填写支票	√	√	√	√	√
显示记录					
库存处理					
获得厂商产品					
维护库存					
处理退单	√	√	√	√	√
审计库存					
调整产品价格					
报表					
创建订单报表					
创建应收账款报表	√	√	√	√	√
创建应付账款报表					
创建库存报表					

下面是另外一些关于回归测试的考虑。

- 当在每个螺旋测试中反复执行时，回归测试将成为测试自动化的一个潜在的候选方案。
- 回归测试应当在系统第一次发布之后进行。
- 发现原始缺陷的测试用例应当在该缺陷被修复后重新执行。
- 应当进一步进行测试以保证不仅仅在表面上修正了原始缺陷。
- 应当去除重复其他测试的回归测试。
- 应当把与发现缺陷相关的功能（或者程序单元）范围内的其他测试用例加入到回归测试集中。

- 客户报告的缺陷应当有很高的优先级并且需要进行彻底的回归测试。

14.1.6 任务 6: 定义测试可交付物

测试的可交付物由测试计划、测试设计、测试开发和测试缺陷文档产生。从螺旋测试中可选择可交付物有如下几种。

- 测试计划: 定义目标、范围、策略、测试类型、测试环境、测试规程、退出标准等 (参见E.4节)。
- 测试设计: 对于应用程序的功能、性能和易用性的测试。证明已经满足原始的测试目标。
- 变更请求: 对当前软件系统进行变更的文档化的要求, 一般由用户提出 (参见附录D以获得更多详细内容)。与报告系统异常的缺陷报告有着显著的不同。
- 度量标准: 系统的一些可量化的方面的测量指标。举例来说, 严重缺陷的数量和发现缺陷的总数可以作为测试人数的函数。
- 测试用例: 为一个特定目标而开发的一系列特定的测试数据和相关流程。这提供了一个具体的蓝图, 指导单个测试的执行, 包括那些特殊的输入数据值和相关期望结果 (参见E.8节以获得更多详细内容)。
- 测试日志总结报告: 将测试人员的个人测试日志中的测试用例按照正在进行或者已完成分类, 为状态报告和度量标准收集做准备 (参见E.10节)。
- 测试用例日志: 记录在测试过程中执行的测试事件中的测试用例。同时也用来记录执行测试的结果, 为测试结果总结提供详细的根据, 如果需要的话, 也为重现测试事件提供基础 (参见E.9节)。
- 中期测试报告: 测试螺旋过程之间发布的报告, 指示测试工作的当前状态 (参见第18章, 步骤3)。
- 系统总结报告: 所有的螺旋测试全部完成之后的一个全面的测试报告 (参见E.11节)。
- 缺陷报告: 记录在螺旋测试过程中发现的缺陷 (参见E.12节)。

173
174

14.1.7 任务 7: 组建测试团队

人员部分包括人力资源分配和需要的相关技能。测试团队应该尽量由才能最高的人员组成。一般来讲由于这些人能力强所以都非常忙, 各个地方都需要他们。因此, 能否按照最佳情况将这些人用于测试变得至关重要。测试团队的领导者和团队本身都需要恰当的技能 and 经验, 并且愿意从事这个项目的的工作。理想情况下, 他们应该都是专业的质量保证专家, 但也能够扮演如下角色: 执行发起人、用户、技术操作、数据库管理、计算机中心、独立第三方等。在任何情况下, 他们都不应该代表开发团队, 因为他们不可能像一个外部单位一样没有偏见。这并不意味着开发人员不能做测试。事实上, 开发人员在将程序提交给测试团队之前, 自己应当做充分的单元测试和功能测试。

在测试中, 有两个职责范围: 测试应用程序, 这是测试团队的责任; 管理测试过程, 这由测试经理负责。测试经理直接管理一名或多名测试人员, 他是质量保证团队和开发团队之间的接口, 管理整个测试工作。其职责包括以下几项。

14

175

- 设定测试目标。
- 定义测试资源。
- 建立测试规程。
- 开发并维护测试计划。
- 设计测试用例。
- 设计并执行自动化测试工具脚本。
- 测试用例开发。
- 提供测试状态。
- 编写报告。
- 定义队伍成员的角色。
- 管理测试资源。
- 定义标准和规程。
- 保证测试过程的质量。
- 培训队伍成员。
- 维护测试统计和度量标准。

测试团队必须由一组有以下职责的队员组成。

- 根据计划执行测试用例。
- 评价测试结果。
- 报告错误。
- 设计并执行自动化测试工具脚本。
- 建议改进应用程序。
- 记录缺陷。

测试团队成员的主要职责是测试应用程序并且将发现的缺陷记录在缺陷跟踪系统中以通知开发团队。一旦开发团队修正了缺陷，测试团队就重新执行发现该缺陷的测试用例。

应当指出，测试经理和测试团队成员的角色并不是互相排斥的。测试经理的一些职责测试团队成员也有，反之亦然。

分配专门的测试资源的依据是测试功能的范围和开发的时间限制。举例来说，一个中等的开发项目比一个小型的项目需要更多的资源。如果一个中等复杂度的项目A需要5个人的测试团队，那么规模是其两倍的项目B就需要10个测试人员（假设测试人员的能力相同）。

另外一个经验是测试的开销应该接近总预算的25%。由于总的项目开销是知道的，可以计算出测试工作量并转化为测试人员的人数。

最好的估算是将项目范围、测试团队技能水平和历史项目综合考虑。对一个特定项目需要多少测试资源进行评估应该依据很多项目的历史资料。也就是说，测试资源的水平和绩效表现应与类似的项目进行比较。

176

14.1.8 任务 8：搭建测试环境

测试环境的目的是为测试活动提供一个必需的物理框架。因此，对本任务而言，应在实现之

前建立并评审测试环境需求。

测试环境的主要组成部分包括物理测试设备、技术和工具。物理测试设备部分包括物理安装, 技术部分包括硬件平台、物理网络及其组件、操作系统软件、其他的一些软件(比如工具软件), 工具部分包括各种专用的测试软件, 比如自动化测试工具、测试库和一些支持软件。

测试设备和工作场所也是需要搭建的。小到个人工作空间的配置, 大到正式的测试实验室。无论如何, 测试人员在一起工作并且与开发团队保持密切联系都是非常重要的。这些设备应能互相通信, 就像有一个共同的目标。同时, 需要的测试工具也应安装。

硬件和软件技术也需要准备。这包括安装测试所需的硬件和软件, 以及协调厂商、用户和信息技术人员。测试硬件和协调厂商也是必需的。通信网络也需要安装和测试。

14.1.9 任务 9: 定义依赖关系

本任务的很好的信息来源是其他项目以前制定的测试计划。如果可以, 项目工作计划中的任务顺序应该按照适用于本项目的活动和任务依赖关系进行分析。

举例来说, 测试依赖关系包括以下几种。

- ☐ 代码可用性。
- ☐ 测试人员可用性(即时方式下)。
- ☐ 测试需求(合理定义的)。
- ☐ 测试工具可用性。
- ☐ 测试组培训。
- ☐ 技术支持。
- ☐ 即时修复缺陷。
- ☐ 充足的测试时间。
- ☐ 计算机和其他一些硬件。
- ☐ 软件和相关文档。
- ☐ 系统文档(如果能得到)。
- ☐ 定义的开发方法。
- ☐ 测试实验室空间可用性。
- ☐ 与开发团队的协议(流程和过程)。

应当定义好相关支持人员并且指派到项目中。这包括开发组成员、技术支持人员、网络支持人员和数据库管理支持人员等。

14.1.10 任务 10: 创建测试进度表

测试进度表应当包含测试步骤(也可能是任务)、目标开始日期和结束日期、职责等, 同时也应该描述出应该如何评审、跟踪和批准该测试进度表。一个简单的测试进度表的格式如表14-2所示, 在螺旋方法之后。

表14-2 测试进度表

测试步骤	开始日期	结束日期	负 责 人
第一次螺旋			
信息收集			
准备访谈	6/1/04	6/2/04	Smith, 测试经理
主持访谈	6/3/04	6/3/04	Smith, 测试经理
总结访谈成果	6/4/04	6/5/04	Smith, 测试经理
测试计划			
建立测试计划	6/8/04	6/12/04	Smith, 测试经理
确定度量目标	6/15/04	6/17/04	Smith, 测试经理
评审/批准计划	6/18/04	6/18/04	Smith, 测试经理
设计测试用例			
设计功能测试	6/19/04	6/23/04	Smith, 测试经理
设计图形用户界面测试	6/24/04	6/26/04	Smith, 测试经理
确定系统/验收测试	6/29/04	6/30/04	Smith, 测试经理
评审/批准设计	7/3/04	7/3/04	Smith, 测试经理
测试开发			
开发测试脚本	7/6/04	7/16/04	Jones、Baker、Brown, 测试人员
评审/批准测试开发	7/17/04	7/17/04	Jones、Baker、Brown, 测试人员
测试执行/评估			
安装及测试	7/20/04	7/24/04	Smith、Jones、Baker、Brown, 测试人员
评估	7/27/04	7/29/04	Smith、Jones、Baker、Brown, 测试人员
为下次螺旋过程做准备			
改进测试	8/3/04	8/5/04	Smith, 测试经理
重新评估团队、规程和测试环境	8/6/04	8/7/04	Smith, 测试经理
发布中期报告	8/10/04	8/11/04	Smith, 测试经理
.			
.			
.			
最后一次螺旋……			
测试执行/评估			
安装及测试	10/5/04	10/9/04	Jones、Baker、Brown, 测试人员
评估	10/12/04	10/14/04	Smith, 测试经理
.			
.			
.			

(续)

测试步骤	开始日期	结束日期	负责人
进行系统测试			
完成系统测试计划	10/19/04	10/21/04	Smith, 测试经理
完成系统测试用例	10/22/04	10/23/04	Smith, 测试经理
评审/批准系统测试	10/26/04	10/30/04	Jones、Baker、Brown, 测试人员
执行系统测试	11/2/04	11/6/04	Jones、Baker、Brown, 测试人员
进行验收测试			
完成验收测试计划	11/9/04	11/10/04	Smith, 测试经理
完成验收测试用例	11/11/04	11/12/04	Smith, 测试经理
评审/批准验收测试计划	11/13/04	11/16/04	Jones、Baker、Brown, 测试人员
执行验收测试	11/17/04	11/20/04	
总结/报告螺旋测试结果			
进行数据归结处理	11/23/04	11/26/04	Smith, 测试经理
准备最终测试报告	11/27/04	11/27/04	Smith, 测试经理
评审/批准最终			
测试报告	11/28/04	11/29/04	Smith, 测试经理; Baylor, 发起人

同时,像Microsoft Project这样的项目管理工具可以将甘特图排版为突出测试的样式,并且按照测试的步骤将测试活动分组。甘特图由任务信息表格和表示测试进度的柱形统计图表组成。同时也包含了任务持续时间,并描述出了任务之间的依赖关系。为了平衡工作量,人力资源可以同样分配到任务中。参见E.13节,另参见模板文件中的甘特螺旋测试方法模板。

另外一个安排测试活动的方法是,通过“相对日程安排法”将测试步骤和任务按照它们的顺序和优先度定义。没有明确的开始日期和结束日期,但是会有一个持续时间,比如多少天。(参见E.18节,测试执行计划用来计划执行阶段的活动;参见E.20节,PDCA测试进度表用来计划和跟踪“计划-执行-检查-改进”测试阶段。)

同时,定义主要的外部 and 内部里程碑也非常重要。外部里程碑指的是一些项目外部的但是对项目有直接影响的事件。比如,项目获得发起人的批准、公司的资金投入以及获得合法的授权。内部里程碑由进度工作计划得来,特别是与需要评审和批准的关键交付文档相关的内容。举例来说,测试计划、设计和开发的完成都需要项目发起人的批准,还有最后一次螺旋测试的总结报告。里程碑可以以表格的形式记录在测试计划中,如表14-3所示。(参见E.19节,测试项目里程碑用来标识和跟踪关键测试里程碑。)

表14-3 项目里程碑

项目里程碑	截止日期
发起人批准	7/1/04
获得第一个原型	7/20/04
项目测试计划	6/18/04
测试开发完成	7/17/04
测试执行开始	7/20/04
发布最后一次螺旋测试的总结报告	11/27/04
系统交付日期	12/1/04

14.1.11 任务 11：选择测试工具

从很简单到非常完善的测试工具软件都存在。同时，为了满足当前应用程序的高软件质量要求，新的工具正在不断地被开发出来。

由于测试工具对测试效率非常重要，因此相关用例的测试执行人应当精通测试工具的使用。要综合考虑测试人员执行操作的环境和将要测试的软件类型，选择最有效的工具。在测试计划中应当指定测试工具及其供应商。选择测试工具的人应同时进行该测试，他应对该工具非常熟悉，能够有效地使用。测试团队应当评审和批准每个测试工具的使用，因为测试工具的选择必须和测试计划的目标相一致。

测试工具的选择可能会基于直觉和判断，但应该更系统化的方法。第六部分将提供获得测试工具的广泛方法，还会提供当前可获得的测试工具类型的概览。

14.1.12 任务 12：建立缺陷报告/跟踪规程

在测试过程中发现的缺陷应当记录下来。每个缺陷都和已经执行的测试相关，目标就是为这些缺陷生成一个完整的记录。记录缺陷的最终动机是为了修正缺陷并且为应用程序提供度量信息。开发人员需要得到缺陷报告，他们将使用缺陷报告来评估是否存在缺陷以及如何修正。缺陷报告的形式可以是手写的也可以是电子版的，我们更加推荐后者。诸如各种类型的缺陷数量和缺陷发现时间这样的度量信息对了解系统的状态是非常有帮助的。

应当建立缺陷控制规程来管理这个过程，从一开始的确认缺陷到最后解决问题。表14-4展现了缺陷从发现到关闭可能经过的一些中间状态。测试部门最初开启一个缺陷报告并且最终关闭它。报告某单元中的一个“是”可能指示了从一个状态到另一个状态的跳转。举例来说，一个“开启”的状态可以转变为“评审中”、“开发人员退回”或者“开发人员延期”。转变可以由开发人员或者测试人员发起。

表14-4 缺陷状态

	开 启	评审中	开发人员返回	等待测试	测试人员返回	开发人员延期	关 闭
开启	—	是	是	—	—	是	—
评审中	—	—	是	是	—	是	是
开发人员返回	—	—	—	—	是	—	是
等待测试	—	—	—	—	是	—	是
测试人员返回	—	—	是	—	—	是	是
开发人员延期	—	是	是	是	—	—	是
关闭	是	—	—	—	—	—	—

缺陷报告的格式同样需要设计。缺陷格式的主要字段包括以下几部分（参见E.12节和E.27节以获得更多详细内容）。

- 问题的确认，比如功能范围、问题类型等。
- 问题的属性，比如行为。
- 导致问题出现的情形，比如输入和步骤。
- 问题出现的环境，比如平台等。
- 诊断信息，比如错误码等。
- 问题的影响，比如因果关系等。

缺陷报告和变更请求的格式很可能是一样的。这种格式相近的好处在于我们并不是每次都清楚变更请求到底是缺陷造成的还是增强的需求引起的。它们之间的区别我们可以通过增加一个格式字段来指示到底是缺陷造成的还是增强的需求。另一方面，在维护阶段，软件的期望行为已经很明确了，可以很容易地区分缺陷和需求增强，这时分开的缺陷报告就很有用。

182
183

14.1.13 任务 13: 建立变更请求规程

如果在系统开发出来之后不会再有任何的变更，那世界就太完美了。但是，在系统被部署完之后，仍然有变更请求。

变更的原因有以下几种。

- 需求变更。
- 设计变更。
- 规约不完整或者有歧义。
- 发现了在评审阶段没有发现的缺陷。
- 软件环境的变更，比如平台、硬件等。

变更控制是对软件组件某个变更进行提出、评估、批准或拒绝、安排进度和跟踪的一个过程。这是控制软件变更的一个决策过程。在这个过程中，可能接受并执行某些提出的变更，也可能拒绝或者延期某些变更。变更控制同样为冲突分析做准备，决定依赖关系（参见附录D以获得更多详细内容。）

每个软件组件都有生命周期。生命周期由一组状态以及状态之间的可能转换构成。只要软件

组件发生了变更，就需要评审。在评审的过程中，需要将组件冻结，不允许再对其做任何更改，想改变它的唯一办法就是建立新的版本。进行评审的人必须决定批准还是拒绝对软件组件的改动。一旦把组件冻结，就要保存到一个软件库中，作为批准组件的存储库。

管理变更的组织正式头衔是配置管理委员会，即CCB（Configuration Control Board）。CCB负责变更的批准工作，决定是否可接受提出的变更。在小型项目中，CCB可能由一个人组成，比如项目经理。在比较正式的开发环境中，CCB可能就会由来自开发部门、用户、质量保证部门、管理部门以及其他相关部门的人共同组成。

所有由软件配置管理控制的组件都存储在软件配置库中，包括工作产品，如业务数据和过程模型、体系结构组、设计单元、经过测试的应用程序软件、可复用的软件和特定的测试软件。当要变更某个组件时，把它从软件配置库中检出，放到一个私有工作空间中。这个组件将进入在配置管理控制范围之外的状态。

当变更完成时，重新把该组件检入软件配置库中，同时产生一个新的组件版本。上一个版本的组件仍然会被保留。

184 变更控制基于以下开发过程来进行：需求分析、系统设计、程序设计、测试和实现。一个变更控制规程至少需要建立6个与以上这些活动相关的控制规程。（参见附录B以获得更多详细内容。）

(1) 初始化规程——这包含通过变更请求表来启动变更请求的规程，该表格是一个交流的工具，目的在于保证变更请求的记录和提交审批的记录是一致的。

(2) 技术评估规程——这包含评估技术上的可行性和风险，以及为一个提出的变更安排技术评估的规程，目的在于保证将提出的变更、测试需求和处理变更请求的能力整合到一起。

(3) 业务评估规程——这包含评估业务风险、影响以及提出的变更的需求落实的规程，目的在于保证提出的变更在时间上与业务目标不冲突。

(4) 管理评审规程——这包含在管理评审会议上对技术评估和业务评估做出评价的规程，目的在于保证变更符合技术和业务的需求并且有充足的与测试和落实相关的资源。

(5) 测试跟踪规程——这包含跟踪并记录测试流程和沟通的规程，包括计划安排测试的步骤、记录测试结果、根据测试结果推迟变更请求以及更新测试日志等，目的在于保证测试标准可以用于验证变更，包括测试计划和测试设计，并且与各方沟通测试结果。

(6) 落实跟踪规程——这包含跟踪和记录变更落实的规程。它保证完成了恰当的批准，分配了充足的时间，定义了落实和备份指令，并且进行了适当的沟通，目的在于保证处理所有已批准的变更，包括安排日期、测试持续时间和测试报告等。

14.1.14 任务 14：建立版本控制规程

唯一标识每个软件组件的方法是制定标识方案。每个软件组件都必须有一个唯一的名称。软件组件的版本将进行不断更新，每一个版本都与其他的不同。为组件定义不同版本的一种简单方法是使用一对整数，比如1.1、1.2等，这定义了发布编号和阶段编号。当第一次标识一个软件组件时，它的版本号是1，后续的主版本号就是2、3等。

185 在客户/服务器架构的环境下，强烈推荐开发环境和测试环境要有所不同。这就要求应用程序的软件组件要从开发环境移植到测试环境，这个规程是需要建立的。

软件应当处于配置管理之下，所以在测试执行过程中不可以发生变更。这包括源文件和可执行组件。应用程序软件可以周期性地移植到测试环境中。必须控制这个过程以保证软件的最后版本是经过测试的。版本控制还帮助管理测试的重复性，保证先前发现的缺陷已经解决。

14.1.15 任务 15: 定义配置构建规程

部署软件系统涉及将源组件或者源代码转化为可执行程序的工具，如编译器、连接编辑程序等。

配置构建规程需要确定正确的组件版本，执行组件构建过程。配置构建模型提出了如何控制组件构建方式的关键问题。

典型的配置由一系列导出的软件组件构成。导出的软件组件的一个例子是来自源程序的可执行目标程序。导出组件必须与每个源组件正确地进行关联以得到正确的出处。配置构建模型提出了如何控制导出组件构建方式的关键问题。

配置构建模型需要的输入和输出包括主要输入和主要输出。主要输入指的是源组件（构建配置的原始资料）、版本选择规程、描述组件之间关系的系统模型。主要输出指的是目标配置和导出的软件组件。

不同的软件配置管理环境使用不同的版本选择方法。最简单的版本选择方法是维护一个组件版本的列表。其他自动方法包括选择最近测试的组件版本或者在特定日期更新的版本等。操作系统实用程序可以用来定义和创建包括目录和命令文件在内的配置。

14.1.16 任务 16: 定义项目问题解决规程

测试问题可能在开发过程的任何时候出现，一旦出现必须成功解决。对问题处理负责的应该是项目经理，他应该与项目发起人一起合作解决这些问题。典型的情况是，测试经理应当记录在测试过程中出现的测试问题，项目经理或者项目发起人应该检查所有发现的问题。可以拒绝或者延期一个问题以做更加深入的调查，但是首先必须考虑这个问题对项目产生的影响。无论如何，都应该生成一个包含基本信息的记录。测试问题的例子有：缺乏测试工具、缺乏充足的测试时间、对需求的认识不充分等。

在项目开始之前首先应该定义好问题管理规程。该规程中应当说明如何进行如下操作。

- 提交问题。
- 报告问题。
- 审阅问题（拒绝、延期、合并或者接受）。
- 调查问题。
- 批准问题。
- 推迟问题。
- 拒绝问题。
- 关闭问题。

14.1.17 任务 17: 建立报告规程

测试报告规程对于管理测试过程和管理项目成员的期望非常重要。这可以帮助项目经理和发

起人时刻了解测试项目的进展并且将不可预期的风险降到最低。测试经理要决定谁需要测试信息，需要什么信息，以及以什么频率向他们提供信息。测试状态报告的目的在于向管理者报告测试进展以及测试的问题、困难和一些相关情况。

下面是两个需要发布的关键报告。

(1) 中期测试报告：中期测试报告是在测试螺旋周期之间发布的、显示测试工作状态的报告。

(2) 系统总结报告：测试总结报告是在所有螺旋测试完成之后的一份全面的测试报告。

14.1.18 任务 18：定义批准规程

批准规程在测试项目中非常重要，有助于在项目参与人员之间建立必须的共识。测试经理要定义谁来对测试的交付物进行审批、何时审批以及如果没有得到批准那么备选方案是什么。批准规程的形式很多，从正式的测试文档的签署到一个非正式的带着评论的评审都有。表14-5展示了测试交付物的批准状态，是必需还是推荐，还有由谁来批准等。（参见E.17节，有一个矩阵可以用来作为测试交付物的正式文档管理批准。）

表14-5 交付物的批准

测试交付物	批准状态	建议审批者
测试计划	必需	项目经理、开发经理、发起人
测试设计	必需	开发经理
变更请求	必需	开发经理
度量标准	推荐	开发经理
测试用例	必需	开发经理
测试日志总结报告	推荐	开发经理
中期测试报告	必需	项目经理、开发经理
系统总结报告	必需	项目经理、开发经理、发起人
缺陷报告	必需	开发经理

14.2 步骤 2：定义度量目标

“你无法控制你不能度量的东西”，这是Tom DeMarco的著作《控制软件项目》一书中的名言，在这本书中他描述了如何组织和控制一个软件项目，使得这个项目在时间和投资规划上都是可度量的。控制是经理保证风险最小化的范围。一旦发现计划偏离应及早报告，这样可以尽快做出响应。DeMarco的书中的另外一句名言，“唯一不可原谅的失败就是没有从过去的失败中吸取教训”，这句话强调了评估和度量的重要性。测量是对过去成果的记录，以便定量地预计将来成果。

14.2.1 任务 1：定义度量标准

作为测试开发项目，软件测试有很多交付物：测试计划、测试设计、测试开发和测试执行等。这个任务的目的是应用度量的原理来控制测试过程。度量标准是系统的某个数量方面的一个可测量的指标，有如下特征。

- 可测量性——从定义上来说,度量点必须是可测量的才可以成为度量标准。如果不可测量,就没有办法使用管理手段来控制它。
- 独立性——度量标准应该独立于人为的影响。除了改变产生该度量标准的事件,没有任何其他办法可以改变该度量标准。
- 可说明性——对原始度量标准数据的任何分析和解释都是建立在该数据本身上的,因此,必须保存好原始数据和分析过程的有条理的审计轨迹。
- 精确性——精确性是精度的一个函数。精确性的关键在于在文档中明确地把度量标准规定为数据收集过程的一部分。如果度量标准是可变的,我们可以当作一个范围来测量。

度量标准可以是一个“结果”或者“预测”。“结果”度量标准用来测量已经完成的事件或者过程。举例来说,执行一个业务事务的实际耗时或者一个项目的总花费。“预测”度量标准是一个早期预警的度量标准,与之后的结果有密切联系。例如,通过统计回归分析,可以在终端数量还没有测量之前,预计出当越来越多的终端加入到系统中时系统的反应时间。“结果”或者“预测”度量标准都可以是导出度量标准。导出度量标准是通过计算或者图示方法、由一个或多个度量标准导出的。

收集测试度量标准的动机是让测试过程更加有效率。这个目标是通过仔细分析度量标准数据,并且采取适当的方法修正出现的问题来实现的。首先要定义度量标准的关注目标。下面是一些例子。

- 缺陷分析——每个缺陷都应该分解为如下问题的回答:根本原因是什么,如何发现的,何时发现的,谁发现的,等等。
- 测试效率——测试进行得如何,例如投资回报率如何?
- 开发效率——开发人员处理缺陷的进度如何?
- 测试自动化——花费了多少精力在测试自动化上?
- 测试成本——在测试方面花费了多少资源和时间?
- 测试状态——另外一个重要的度量标准是状态跟踪,或者说现在处于测试过程的什么阶段?
- 用户参与——有多少用户参与到了测试中?

14.2.2 任务 2: 定义度量要点

表14-6中列出了一些度量要点,这些要点有的和上一个任务中选择的一般度量有关,有些和能够改进测试过程的行为有关。同时也展示出了度量要点的来源或者出处。

表14-6 度量要点

度量标准	度量要点	来源
缺陷分析	缺陷起因的分布	直方图, 帕累托图
	按时间统计的不同原因的缺陷数目	多线图
	按时间统计的不同发现方法的缺陷数目	多线图
	不同模块的缺陷分布	直方图, 帕累托图

(续)

度量标准	度量要点	来 源
缺陷分析	不同优先级的缺陷分布 (紧急、高、中、低)	直方图
	不同功能区域的缺陷分布	直方图
	不同环境 (平台) 的缺陷分布	直方图, 帕累托图
	不同类型的缺陷分布 (架构、连接、一致性、数据库完整性、文档、图形用户界面、安装、内存、性能、安全性、标准和惯例、压力、易用性、错误的修复)	直方图, 帕累托图
	不同发现人员的缺陷分布 (外部用户、内部用户、开发人员、质量保证人员及其他人员)	直方图, 帕累托图
	不同发现方式的缺陷分布 (技术评审、走查、JAD、原型、审查、测试执行)	直方图, 帕累托图
	不同重要性的缺陷分布 (高、中、低)	直方图
开发效率	开发人员修复缺陷的平均时间	总的修复时间 ÷ 修复的缺陷数量
测试自动化	手工测试和自动化测试的比例	手工测试工作量 ÷ 总的测试工作量
测试成本	不同原因的成本分布	直方图, 帕累托图
	不同应用程序的成本分布	直方图, 帕累托图
	测试占总成本的比例	测试成本 ÷ 系统总成本
	按时间统计的测试总成本	线图
	定位一个缺陷的平均成本	测试总成本 ÷ 发现的缺陷数目
	预期测试成本和实际成本的比较	对照
	通过需求评审定位一个需求缺陷的平均成本	需求评审成本 ÷ 需求评审阶段发现的缺陷数目
	通过设计评审定位一个设计缺陷的平均成本	设计评审成本 ÷ 设计评审阶段发现的缺陷数目
	通过代码评审定位一个代码缺陷的平均成本	代码评审成本 ÷ 代码评审阶段发现的缺陷数目
	通过测试执行定位一个缺陷的平均成本	测试执行成本 ÷ 测试执行阶段发现的缺陷数目
	按时间统计的测试资源数目	线图
测试效率	维护过程中发现的缺陷的百分比	维护过程中发现的缺陷数目 ÷ 发现的缺陷总数
	测试发现缺陷的百分比	测试过程中发现的缺陷数目 ÷ 发现的系统缺陷总数
	测试的平均效率	测试数量 / 发现的系统缺陷总数
	需求评审回报价值	需求评审过程中发现的缺陷数目 ÷ 需求测试成本
	设计评审回报价值	设计评审过程中发现的缺陷数目 ÷ 设计测试成本
	程序评审回报价值	程序评审过程中发现的缺陷数目 ÷ 程序测试成本
	测试执行回报价值	测试执行过程中发现的缺陷数目 ÷ 测试成本
	测试变更的效果	测试变更的数目 ÷ 可归于变更的问题数量
	人们对于测试效率的评价	主观评价 (1~10)

(续)

度量标准	度量要点	来 源
测试效率	质量保证人员检验修正的平均时间	总的测试人员检验时间÷需要检验的缺陷总数
	按时间统计的缺陷数量	线图
	按时间统计的缺陷累计数量	线图
	按时间统计的不同应用程序缺陷数量	多线图
测试范围	语句执行百分比	执行的语句数目÷语句总数
	逻辑路径执行百分比	执行的逻辑路径数目÷逻辑路径总数
	验收标准经过测试的百分比	经过测试的验收标准÷验收标准总数
	按时间统计的需求测试数量	线图
	按时间统计的语句执行数量	线图
	按时间统计的数据元素使用数量	线图
	按时间统计的判断语句执行数量	线图
测试状态	按时间统计的等待运行的测试数量	线图
	按时间统计的测试运行数量	线图
	没有发现缺陷的测试运行数量	线图
	按时间统计的被修正的缺陷数量	线图
用户参与	用户测试的百分比	用户测试时间÷测试总时间

193

14.3 步骤 3: 测试计划的评审和批准

14.3.1 任务 1: 评审的日程安排/执行

在实际进行评审之前,应该首先安排好测试计划评审的时间,同时保证所有参与人员都拿到测试计划的最新版本。

像其他评审一样,测试计划的评审应该包含如下几个必需的部分。首先要定义讨论的内容(即“我们讨论将讨论什么”),第二步要讨论细节问题(即“我们正在讨论”),第三步是总结(即“我们讨论了什么”),最后是时间安排。评审人员应该预先规定评审的预计持续时间,并且制定一条基本原则,如果规定的时间到了而议程表中的项目没有全部讨论完,就需要再安排一次后续的评审。

本项任务的目的是让开发人员和项目发起人都能同意并接受测试计划。如果评审过程中有任何对测试计划的改进建议,都应合并到测试计划中去。

14.3.2 任务 2: 获得批准

获得批准在测试工作中是非常重要的,因为这可以保证测试人员、开发人员和项目发起人三者之间达成必要的、一致的协定。获得批准的最佳方法是通过正式的测试计划签署手续,如果有

的话,使用管理审批签署表。然而,如果没有正式的一致承认的手续,则向每位关键的参与人员发送一份备忘录,这些人员至少包括项目经理、开发经理和项目发起人。在文档中要附上最新的测试计划并指出他们反馈的所有意见都被合并进来了,而且如果没有得到反馈,则默认他们同意这一测试计划。最后,要指出在螺旋式开发环境中,测试计划是会随着每次迭代过程而演进的,我们会把任何修正都包含进来。

回忆一下，在螺旋式开发环境中，软件测试可以描述为必须集成到快速应用开发方法中的持续改进过程。应用PDCA模型的Deming持续改进过程也将应用到软件测试过程中。我们现在探讨螺旋式模型的“执行”部分（见图15-1）。

图15-2列出了与螺旋测试“执行”部分相关联的步骤和任务。在描述每一个步骤和每一项任务的同时，也给出了一些有价值的提示和技术。



图15-1 螺旋测试和持续改进

15.1 步骤 1：设计功能测试

15.1.1 任务 1：完善功能测试需求

此时，功能规约应当已经完成了。它由以下部分组成：分层的功能分解、功能窗口结构、窗口标准以及要开发的系统的最低系统需求。举例来说，窗口标准可以是Windows 2000图形用户界面标准，最低系统需求可能由以下部分组成：Windows 2000、Pentium IV微处理器、

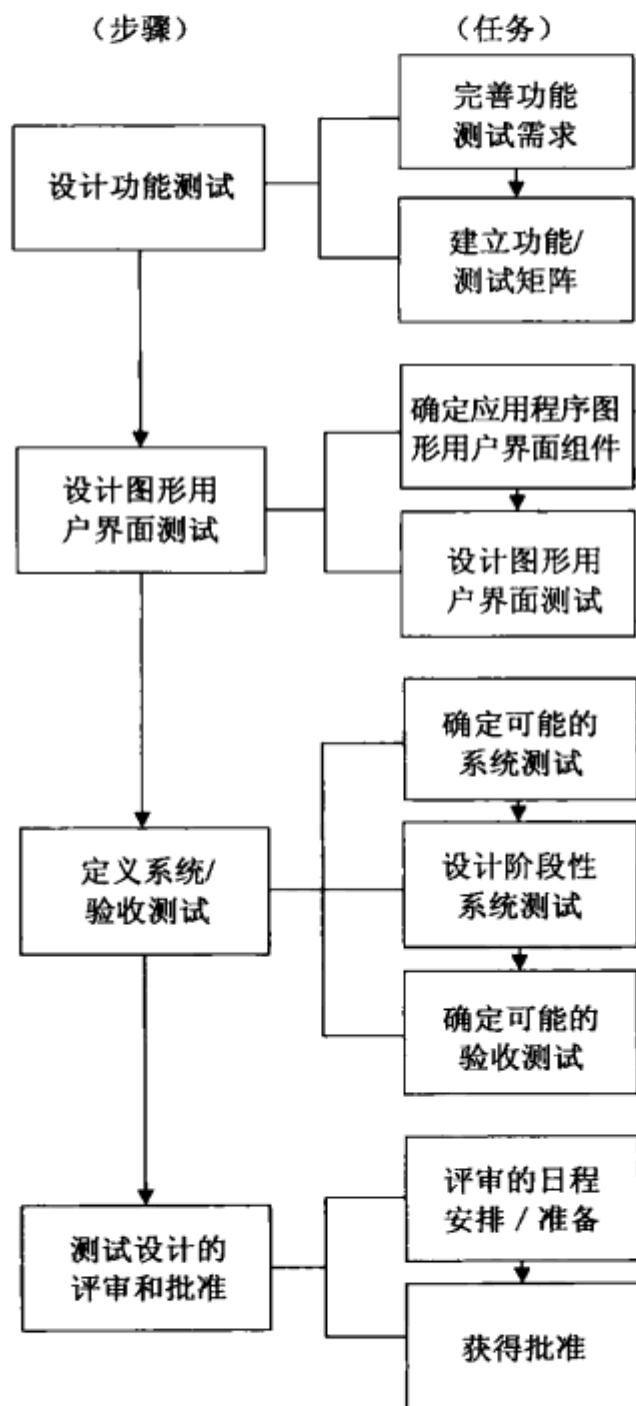


图15-2 测试用例设计（步骤/任务）

1 GB的内存、40 GB的硬盘空间和56 KB的调制解调器。

功能分解包括业务功能的列表、分层的清单或者一系列使用剖面图定义出系统的基本功能以及用户如何使用。业务功能是业务的一个非连续的可控制方面，也是系统的最小组件。每一个业务功能都应该按“动宾”结构命名和描述。还应当给出判断一项功能是否正确执行的标准。功能的分层是功能测试的基础，它为每个最底层的功能都至少有一个对应的测试用例提供了保证。功能的例子包括批准客户信用、处理订单、打印发票、订单构成、收到款项、支付账单、购买商品等。总的来说，业务功能组成了包含任意接口的整个应用程序。这些功能的比较好的来源是（除了会议之外）过程分解或者数据流图，或者在信息收集会议上需要的CRUD矩阵。

需求是创建测试用例的基础。以下质量保证测试检查表可以用来保证需求的清晰和完整。

- 澄清请求（E.22节），用于记录测试人员在分析需求时可能出现的问题（E.22节）。
- 歧义性评审检查表（F.25节），用于辅助为了发现一些结构歧义性而对功能规约进行的评审（不要与内容评审混淆）。
- 架构评审检查表（F.26节），用于评审架构的完整性和清晰性。
- 数据设计评审检查表（F.27节），用于评审逻辑设计和物理设计内容的完整性和清晰性。
- 功能规约评审检查表（F.28节），用于评审功能规约内容的完整性和清晰性（不要与歧义性评审混淆）。
- 原型评审检查表（F.29节），用于评审原型内容的完整性和清晰性。
- 需求评审检查表（F.30节），用于验证测试项目需求的全面性和完整性。
- 技术设计评审检查表（F.31节），用于评审技术设计的完整性和清晰性。

功能分解用来说明分层结构下的各个过程，展示了各个连续层次的详细信息。它通过普通过程和特殊过程的分解而迭代构成（见图15-3）。

数据流图展示了很多过程和这些过程之间的数据流向。数据流图用来定义一个系统的全部数据流动，它由外部代理组成，外部代理与系统、过程、数据流、存储与检索数据的存储描述接口。数据流图也需要评审，每个主要的、分层的功能都应该被组织和列入到一个分层次的列表中。

CRUD矩阵，或者关联矩阵，将数据模型和过程模型联系起来。它标识并解决了矩阵冗长和冲突问题，有助于完善数据模型和过程模型。

功能窗口结构描述了在窗口环境中功能应如何实现。图15-4展示了一个订单处理功能窗口结构的例子。

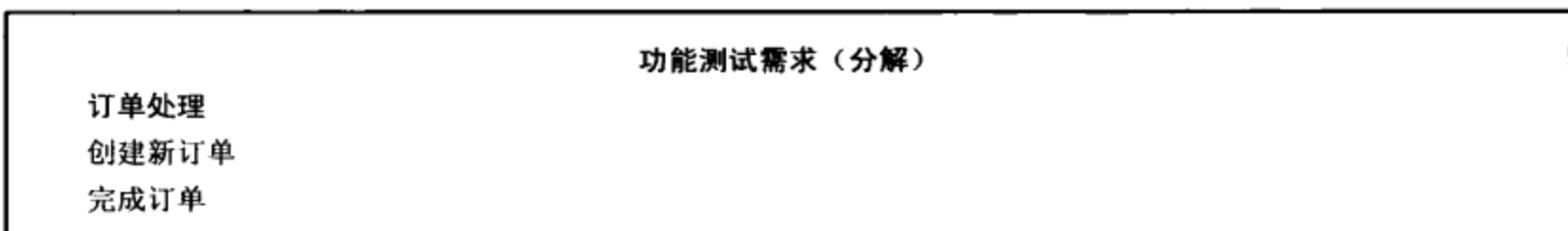


图15-3 功能分解

编辑订单
删除订单
客户处理
创建新客户
编辑客户
删除客户
财务处理
接受客户付款
存储付款
支付供应商
填写支票
显示记录
库存处理
获得供应商产品
维护库存
处理退单
审计库存
调整产品价格
报表
创建订单报表
创建应收账款报表
创建应付账款报表
创建库存报表

图15-3 （续）

198

主窗口
a. 主窗口的第一行是标准的标题栏和最小化/最大化控制按钮。
b. 下一行是标准的窗口菜单栏。
c. 再下一行是标准的窗口工具栏。
d. 主应用程序窗口的其余空间是客户订单窗口。
客户订单窗口
a. 这个窗口中要显示一个先前输入的所有订单的摘要。
b. 一次显示几个订单（用订单编号和客户名称分类）。每个客户订单在这个窗口中要显示如下内容：

图15-4 功能窗口结构

15

1. 订单编号
 2. 客户姓名
 3. 客户编号
 4. 日期
 5. 发票编号
 6. 型号编号
 7. 产品编号
 8. 装运数量
 9. 价格
- c. 使用滚动条来帮助选择想详细查看的订单。
 - d. 对于查看来讲本窗口是只读的。
 - e. 双击一个订单将打开能够修改这个订单的编辑订单对话框。

编辑订单窗口

- a. 这个对话框用来创建新的订单或者对已创建的订单进行修改。
- b. 这个对话框会处于客户订单窗口的中央。此对话框的布局如下：
 1. 订单编号 (自动填入)
 2. 可编辑区域：客户姓名
 3. 可编辑区域：客户编号
 4. 日期 (已初始化的)
 5. 可编辑区域：发票编号
 6. 可编辑区域：型号编号
 7. 可编辑区域：产品编号
 8. 可编辑区域：装运数量
 9. 价格 (自动填入)
 10. 按钮：确定和取消

菜单栏将包含下列菜单

文件

- 新建：新建一个订单文件
- 打开：打开一个订单文件
- 保存：保存一个订单文件
- 另存为：将当前的订单文件保存为一个新的名字
- 退出：退出窗口

订单

- 建立新的订单：显示带有空白区域 (日期除外) 的编辑订单窗口
- 完成订单：这个窗口用来验证库存是否可以满足订单数量并且确认客户信用。这个对话框包含如下内容：
 1. 可编辑区域：订单编号
 2. 可编辑区域：客户姓名
 3. 可编辑区域：客户编号
 4. 日期 (已初始化的)

图15-4 (续)

5. 发票编号 (已初始化的)
6. 型号编号 (已初始化的)
7. 产品编号 (已初始化的)
8. 装运数量 (已初始化的)
9. 价格 (已初始化的)
10. 按钮: 确定和取消
 - a. 这些订单已经与库存核对过。如果订单不能执行, 将给购买者发延期交货通知
 - b. 显示客户历史记录 (使用滚动条来查看历史信息)
 - c. 单击接受按钮将完成这个订单并且创建一个供装运使用的发票
 - d. 单击拒绝按钮将删除该订单并且创建一封拒绝客户订单的信

编辑订单: 这个对话框用来编辑已经存在的订单。这个对话框包含如下内容:

1. 可编辑区域: 订单编号
2. 可编辑区域: 客户姓名
3. 可编辑区域: 客户编号
4. 按钮: 确定和取消

删除订单: 这个对话框用来删除已经存在的订单。这个对话框包含如下内容:

1. 可编辑区域: 订单编号
2. 可编辑区域: 客户姓名
3. 可编辑区域: 客户编号
4. 按钮: 确定和取消
 - a. 选择是、否或者取消选项后都会出现一个请求确认的消息

订单报表: 这个对话框将根据订单编号或者日期范围显示一个或多个订单。这个对话框的布局如下:

1. 单选按钮: 订单, 日期
2. 第一个订单编号 (按照订单生成报表)
3. 备选的最后订单编号 (按照订单生成报表)
4. 最早的日期 (按照日期生成报表)
5. 备选的最晚日期 (按照日期生成报表)

在每份报表后用户将会看到如下提示信息: “您需要打印吗?”

用户将会看到如下提示信息: “您需要另外一份报表吗 (是/否)? ”

视图

工具栏: 使工具栏在显示和隐藏两种状态间切换

状态栏: 使状态栏在显示和隐藏两种状态间切换

工具栏使用图标来执行下列菜单指令

文件→新建
 文件→打开
 订单→新建
 订单→确认
 订单→编辑
 订单→删除
 文件→退出

图15-4 (续)

15.1.2 任务2：建立功能/测试矩阵

功能/测试矩阵将测试和功能关联起来。这个矩阵提供了证明测试策略完整性的方法，以图解的形式说明了哪些测试用例测试哪些功能。（如想了解更多信息，请参见表15-1和E.5节的功能/测试矩阵。）

表15-1 功能/测试矩阵

业务功能	测试用例				
	1	2	3	4	5
订单处理					
创建新订单	CNO01	CNO02			
完成订单	AO01				
编辑订单	EO01	EO02	EO03	EO04	
删除订单	DO01	DO02	DO03	DO04	DO05
客户处理					
创建新客户	ANC01	ANC02	ANC03		
编辑客户	EC01	EC02	EC03	EC04	EC05
删除客户	DC01	DC02			
财务处理					
接收客户付款	RCP01	RCP02	RCP03	RCP04	
存储付款	AP01	AP02			
支付厂商	PV01	PV02	PV03	PV04	PV05
填写支票	WC01	WC02			
显示记录	DR01	DR02			
库存处理					
获得厂商产品	AP01	AP02	AP03		
维护库存	MS01	MS02	MS03	MS04	MS05
处理退单	HB01	HB02	HB03		
审计库存	AI01	AI02	AI03	AI04	
调整产品价格	AC01	AC02	AC03		
报表					
创建订单报表	CO01	CO02	CO03	CO04	CO05
创建应收账款报表	CA01	CA02	CA03		
创建应付账款报表	AY01	AY02	AY03		
创建库存报表	CI01	CI02	CI03	CI04	

这个矩阵是测试进行时的一个控制表，同样也可以用在维护阶段。举例来说，如果要变更一个功能，维护团队就可以查找功能/测试矩阵，确定哪些测试需要重复运行或者变更。业务功能

在垂直方向列出, 测试用例在水平方向列出。测试用例名称连同编号一起记录在矩阵上。(另见 E.24 节, 这个矩阵用于将需求与映射到一个或多个测试用例的每个条件相关联。)

区分手动执行和自动执行的测试用例也十分重要。实现这个目标的方式是提出一种强调自动化的测试用例的命名标准; 举例来说, 名字的第一个字母定义为“A”。

表15-1展示了一个功能/测试矩阵的例子。

15.2 步骤 2: 设计图形用户界面测试

图形用户界面 (GUI) 设计的目标是为应用程序用户实现比较好的界面风格。好的图形用户界面有两个关键的部分: 交互性和外观。交互性关心用户如何与应用程序进行交互, 外观则关心用户对界面的感觉如何。

图形用户界面测试主要包括确认“导航”是否正确。举例来说, 当点击一个图标、菜单选项或者单选按钮时, 应当出现我们期望的响应。下面是一些图形用户界面的设计原则, 测试人员在测试应用程序时可以适当参照。

图形用户界面设计的10条指导原则

- (1) 让用户参与进来。
- (2) 了解用户的文化修养和经验。
- (3) 不断更改原型来验证需求。
- (4) 以用户的业务工作流程来驱动设计。
- (5) 不要过分使用, 也不要浪费图形用户界面的特性。
- (6) 同时创建图形用户界面、帮助文件和培训文件。
- (7) 不要期望用户能够记住指令和函数。
- (8) 预期使用中可能出现的错误, 不要给出不友好的提示。
- (9) 持续提醒用户应用程序的当前状态。
- (10) 尽量简洁。

200
201

15.2.1 任务 1: 确定应用程序图形用户界面组件

图形用户界面提供了多种交流的渠道: 文字、图片、动画、语音、视频等。用户界面的5个基础组件: 窗口、菜单、表单、图标和控件等。

(1) 窗口: 在窗口环境中, 用户和应用程序的所有交互动作都是在窗口中进行的。这些窗口包括一个主窗口和由它生成的任意数量的二级窗口。

(2) 菜单: 菜单有多种多样的格式和类型。举例来说, 动作菜单 (按钮、单选按钮)、下拉菜单、弹出菜单、选项菜单、层叠式菜单等。

(3) 表单: 表单是可供用户填写信息的窗口或区域。

(4) 图标: 图标, 或者“可视按钮”, 对于快速识别很有价值, 容易学会使用, 容易在应用程序中做导航。

(5) 控件: 界面上的控件使得用户可以与应用程序进行交互, 并且标出该控件所代表的功能。

控件包括菜单栏、下拉菜单、层叠式菜单、弹出菜单、按钮、复选框、单选按钮、列表框和下拉列表框等。

进行图形用户界面设计的第一步就是根据应用程序中的名称对每个图形用户界面组件进行定义和命名，如表15-2所示。在下一步中，将生成一个对这一表格中的每个组件进行确认的图形用户界面组件检查表（参见E.6节）。

表15-2 图形用户界面测试矩阵

名 称	图形用户界面类型					通过/失败	日 期	测 试 者
	窗 口	菜 单	表 单	图 标	控 件			
主窗口	√							
客户订单窗口	√							
编辑订单窗口	√							
菜单栏		√						
工具栏					√			

15.2.2 任务 2：设计图形用户界面测试

在前一项任务中，我们对应用程序图形用户界面的组件进行了定义、命名，并且将它们分类填入到图形用户界面组件测试矩阵中。在当前任务中，我们将创建一个用来对每个图形用户界面组件进行验证的检查表。这个检查表应当覆盖所有的交互活动，但对于某个特定的组件可能适用也可能不适用。表15-3所示是一个要检验的项目的部分检查表。（参见E.23节，屏幕数据映射可以用来列出屏幕数据的属性；参见F.32节，测试用例准备工作评审检查表可以用来确保对于每个具体的点都准备了测试用例。）

表15-3 图形用户界面组件检查表

通过双击访问	多窗口打开	标签顺序
通过菜单访问	Ctrl菜单（移动）	按钮
通过工具栏访问	Ctrl + 功能键	下拉菜单和子菜单选项
右键选项	颜色	对话框控件
帮助链接	快捷键和热键	标签
上下文相关的帮助	取消	袖章
按钮栏	关闭	椭圆
双击打开	应用	外部灰色的代表不可用
屏幕图像和图画	退出	复选框
菜单打开	确定	过滤器
工具栏打开	水平/垂直排布	旋转盒
图标访问	范围图标	滚动条
访问DOS	锁定	字体

(续)

通过双击访问	多窗口打开	标签顺序
通过单击使用	发散/聚集树	拖/放
调整窗口面板大小	功能键	水平/垂直卷动
接受允许数据的字段	窗口最小化 窗口最大化	层叠式窗口打开
处理无效数据的字段	标签顺序	

除了对以上图形用户界面组件进行检查以外,如果有图形用户界面设计标准的话,也要检查该标准。图形用户界面标准的本质是要保证设计遵循了内部构造规则以使一致性达到期望的水平。下面是一些需要验证的典型图形用户界面标准。

- 表单“可输入”和只读格式。
- 提示、错误消息和帮助特性。
- 颜色、高亮和光标的使用。
- 屏幕的布局。
- 功能键和快捷键,即“热键”。
- 屏幕上元素排布的一致性。
- 对象的逻辑顺序。
- 字体使用的一致性。
- 颜色使用的一致性。

同时,区分手工的和自动的图形用户界面测试用例也很重要。可以在图形用户界面组件矩阵中增加一列,该列中的值指出对应行的图形用户界面测试用例是手工的还是自动的。

15.3 步骤 3: 定义系统/验收测试

15.3.1 任务 1: 确定可能的系统测试

系统测试是最高级别的测试,从整个系统的角度评价功能和性能以及整体的适用性。这种测试通常由内部组织完成,是一个面向系统技术问题的测试,而不是像验收测试那样是用户导向的测试。

系统测试由一个或多个基于系统原始目标的测试组成,这些原始目标是指在项目访谈会议上定义的那些目标。本项任务的目标是选择将要进行的系统测试,而不是如何实现这些测试。下面是一些常见的系统测试类型。

- 性能测试——验证并且确认达到了性能需求,测量响应时间、事务处理效率及其他一些与时间相关的需求。
- 安全性测试——评估应用程序的安全性,保证数据的完整性和机密性。
- 容量测试——对应用程序进行大数据量的测试,确定应用程序是否能够处理一定数量的数据。

- 压力测试——研究系统在资源超载情况下的表现。尤为关心的是对于系统处理时间的影响。
- 兼容性测试——测试应用程序与其他应用程序或系统的兼容性。
- 转换测试——验证转换当前数据并加载到一个新的数据库的情况。
- 易用性测试——测定用户对应用程序的使用及理解程度。
- 文档测试——验证用户文档的正确性并且确保手册中描述的规程可以走通。
- 备份测试——验证当系统遇到软件或者硬件失败时，备份数据的能力。
- 可恢复性测试——验证系统从一次软件或者硬件失败中恢复的能力。
- 安装测试——验证成功安装系统的能力。

15.3.2 任务2：设计阶段性系统测试

阶段性系统测试是在每个螺旋循环中都要执行的整个系统测试的一个样本子集。进行阶段性测试的目的在于为将在整个系统测试时可能出现的尚未解决的问题提供一个早期的警告。可选的阶段性系统测试包括功能测试、性能测试、安全性测试、易用性测试、文档测试和规程测试。这些阶段性测试中的一部分需要在每个螺旋过程中都进行正式的测试，而其他的则是整体测试策略的一部分。不能分阶段的系统测试包括安装测试、恢复性测试、转换测试等，这些测试要等到正式的系统测试时才能进行。

当系统在每个螺旋中集成起来的时候，系统级的功能测试就可以执行了。在每个螺旋中，如果有新的功能添加，就要设计、实现并执行新的测试用例。

通常，在开发初期就引进了安全性机制。因此，在每个螺旋周期中，随着越来越多的系统功能添加进来，要设计、实现并执行一组安全性测试用例。

易用性测试是在每个螺旋周期中都要进行的非正式测试，应当始终是测试策略的一部分。当发现易用性缺陷时，测试人员应当将其记录在缺陷跟踪系统中。易用性测试的正式方式是由终端用户在每个螺旋周期对原型系统进行的评审。

文档测试（如在线帮助）和规程测试同样也是在每个螺旋周期都要进行的非正式测试。在每个螺旋周期中，这两种测试应当与正式的系统开发保持同步，而不要一直拖延到正式的系统测试。这样做可以避免最后一刻出现危机（如出现大量错误）。在每个螺旋周期中，随着越来越多系统功能的增加，我们要设计、实现并测试新的文档和规程。

一些单用户性能测试应当在每个螺旋周期中都进行。每个关键功能添加到系统中时都应进行基线测量。基线测量的目的是为状态或者性能测量提供一个初始值。在后面的螺旋周期中，将重复执行性能测量并且与基线进行比较。表15-4提供了一个基线性能测量的例子。

表15-4 基线性能测量

商业功能	基线时间（秒） Rel 1.0 (1/1/2004)	测量时间（秒） Rel1.1 (2/1/2004)	测量时间（秒） Rel1.2 (2/15/2004)	测量时间（秒） Rel1.3 (3/1/2004)	测量时间（秒） Rel1.4 (3/15/2004)	测量时间（秒） Rel1.5 (4/1/2004)
订单处理						
创建新订单	1.0 (+50%)	1.5 (-13%)	1.3 (-23%)	1.0 (-10%)	0.9 (-17%)	0.75

(续)

商业功能	基线时间 (秒) Rel 1.0 (1/1/2004)	测量时间 (秒) Rel1.1 (2/1/2004)	测量时间 (秒) Rel1.2 (2/15/2004)	测量时间 (秒) Rel1.3 (3/1/2004)	测量时间 (秒) Rel1.4 (3/15/2004)	测量时间 (秒) Rel1.5 (4/1/2004)
完成订单	2.5 (-20%)	2.0 (-25%)	1.5 (-33%)	1.0 (0%)	1.0 (0%)	1.0
编辑订单	1.76 (+14%)	2.0 (+25%)	2.5 (-32%)	1.7 (-12%)	1.5 (-20%)	1.2
删除订单	1.1 (0%)	1.1 (+27%)	1.4 (-29%)	1.0 (-20%)	0.8 (-6%)	0.75
.
.
.
.
.

15.3.3 任务 3: 确定可能的验收测试

验收测试是一种由用户执行的可选测试, 用来证明应用程序具备满足用户需求的能力。这个测试的出发点是证明功能而不是发现错误, 也就是要证明这个系统能够工作。测试的重点少部分在技术问题上, 大部分在“对于最终用户来说, 系统是否具有较好的业务适用性”这个问题上。如果需要进行验收测试, 一般由用户来完成。通常, 20%的情况下这个测试是集成在系统测试中进行的。如果执行的话, 一般来说, 验收测试是系统测试的一个子集。然而, 用户有时会定义一些“特殊的测试”, 比如高强度压力测试或者容量测试, 挑战系统的极限, 甚至有时超过系统测试时所测试的。

15.4 步骤 4: 测试设计的评审和批准

15.4.1 任务 1: 评审的日程安排/准备

在实际进行评审之前, 应该首先安排好测试设计评审的时间, 同时保证所有参与人员都有测试设计的最新版本。

像其他访谈和评审一样, 测试设计的评审应该包含如下几个必需的部分。首先要定义讨论的内容 (即“我们将要讨论什么”), 第二步要讨论细节问题 (即“我们正在讨论”), 第三步是总结 (即“我们讨论了什么”), 最后是时间安排。评审人员应该预先规定评审的预计持续时间, 并且制定一条基本原则, 如果规定的时间到了而议程表中的项目没有全部讨论完, 就需要再安排一次后续的评审。

本项任务的目的是让开发人员和项目发起人都能同意并接受测试设计。如果评审过程中有任何对测试设计的改进建议, 都应合并到设计中去。

15.4.2 任务 2: 获得批准

获得批准在测试工作中是非常重要的,因为这可以保证测试人员、开发人员和项目发起人三者之间达成必要的、一致的协定。获得批准的最佳方法是通过正式的测试设计签署手续,如果有的话,使用管理审批签署表。然而,如果没有正式的一致承认的手续,则向每位关键的参与人员发送一份备忘录,这些人员至少包括项目经理、开发经理和项目发起人。在文档中要附上最新的测试设计并指出他们反馈的所有意见都合并进来了,而且如果没有得到反馈,则默认他们同意这一设计。最后,要指出在螺旋式开发环境中,测试设计会随着每次迭代过程而演进,我们会把任何修正都包含进来。

图16-1列出了与螺旋测试的“执行”部分相关联的步骤和任务。在描述了每一个步骤和每一项任务的同时，也给出了一些有价值的提示和技术。

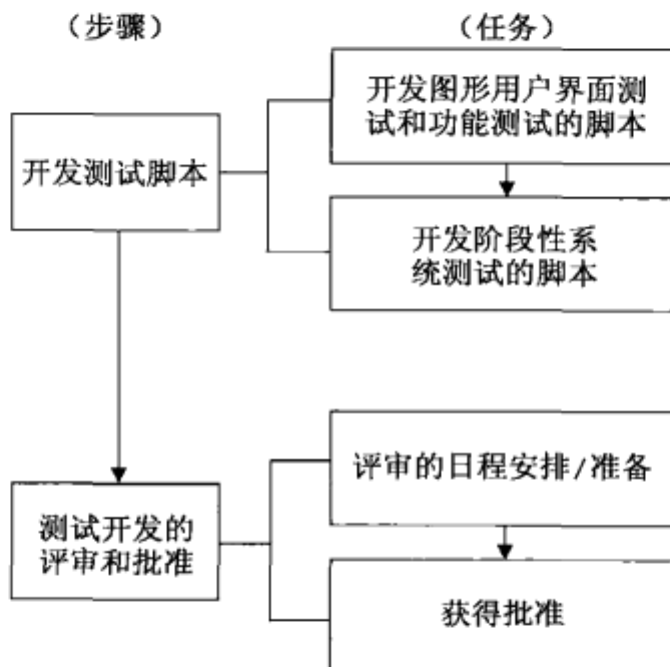


图16-1 测试开发（步骤/任务）

16.1 步骤 1：开发测试脚本

16.1.1 任务 1：开发手工/自动化图形用户界面/功能测试脚本

在第15章，已经建立了将测试与功能关联起来的图形用户界面/功能测试矩阵。业务功能垂直列出，测试用例水平列出。测试用例的名称连同编号一起记录在矩阵中。

在本项任务中，把功能测试用例写到专门的文档中，并通过创建的测试数据转换成可复用的测试脚本。为了辅助开发测试用例脚本，表16-1中的基于图形用户界面的功能测试矩阵模板可以用于记录基于用户图形界面的功能测试用例（参见E.7节基于图形用户界面的功能测试矩阵可得到更加详细的信息）。

表16-1展示脚本使用模板创建了新客户订单。模板的使用展示了测试的功能、测试用例中的用例序号、需求交叉索引标识、测试目标、用例步骤、期望结果、测试结果（通过/失败状态）、

测试者姓名以及进行测试的日期。在一个功能中，当前使用的图形用户界面组件也被列出。例如在表16-1中，一个新客户订单的创建首先要调用菜单栏来选择该功能，随后在订单编辑窗口中输入订单编号、客户编号、型号编号、产品编号和订购数量。

表16-1 功能测试和图形界面测试脚本

功能（创建一个新的客户订单）							
用例编号	需求编号	测试目标	用例步骤	期望结果	测试结果	测试者	测试日期
菜单栏							
15	67	创建一个有效的新客户订单	在菜单栏中选择“文件/创建订单”	弹出订单编辑窗口	通过	Jones	7/21/2004
编辑订单窗口							
			1. 输入订单编号	订单确认	通过	Jones	7/21/2004
			2. 输入客户编号	客户确认	通过	Jones	7/21/2004
			3. 输入型号编号	型号确认	通过	Jones	7/21/2004
			4. 输入产品编号	产品确认	通过	Jones	7/21/2004
			5. 输入订购数量	数量确认，生成日期、发票号和总价格	通过	Jones	7/21/2004
			6. 单击“确定”按钮	创建客户订单成功	通过	Jones	7/21/2004

16.1.2 任务2：开发手工/自动化阶段性系统测试脚本

在前面的任务中设计了阶段性系统测试。阶段性系统测试是全面系统测试中抽样出来的有代表性的子集，可以在每次螺旋循环中都执行。

在本项任务中，可以利用上一个任务中讨论的基于图形用户界面的功能测试矩阵来创建阶段性系统测试的脚本。测试目标的描述应该比功能/图形用户界面测试范围更广一些，因为要包括更多的整体测试问题，比如性能、安全性、易用性、文档、规程等。

16.2 步骤2：测试开发的评审和批准

16.2.1 任务1：评审的日程安排/准备

在实际进行评审之前，应该首先安排好测试开发评审的时间，同时保证所有参与人员都有测试开发的最新版本。

像其他访谈和评审一样，测试开发的评审应该包含如下几个必需的部分。首先要定义讨论的内容（即“我们将要讨论什么”），第二步要讨论细节问题（即“我们正在讨论”），第三步是总结（即“讨论我们讨论了什么”），最后是时间安排。评审人员应该预先规定评审的预计持续时间，

并且制定一条基本原则,如果规定的时间到了而议程表中的项目没有全部讨论完,就需要再安排一次后续的评审。

本项任务的目的是让开发人员和项目发起人都能同意并接受测试开发。如果评审过程中有任何对测试开发的改进建议,都应合并到测试开发中去。

16.2.2 任务 2: 获得批准

获得批准在测试工作中是非常重要的,因为这可以保证测试人员、开发人员和项目发起人三者之间达成必要的、一致的协定。获得批准的最佳方法是通过正式的测试开发签署手续,如果有的话,使用管理审批签署表。然而,如果没有正式的一致承认的手续,则向每位关键的参与人员发送一份备忘录,这些人员至少包括项目经理、开发经理和项目发起人。在文档中要附上最新的测试开发并指出他们反馈的所有意见都被合并进来了,而且如果没有得到反馈,则默认他们同意这一测试开发。最后,要指出在螺旋式开发环境中,测试开发会随着每次迭代过程而演进,我们会把任何修正都包含进来。

通过可追溯性实现测试覆盖

大多数企业在软件稳定之前都能容忍其存在一定数量的缺陷。但是，在严重缺陷没有解决的情况下系统不可能运行良好。多数公司会在测试策略文档中说明对软件的验收标准，验收标准的范围可能从不存在严重和中等级别的缺陷到业务流程可以被终端用户接受。最后测试的终极目标就是证明软件实现了客户的所有需求。不同的测试交付物之间的联系要保证测试全面覆盖了所有的需求，也就是说所有的需求都被测试到了，没有任何遗漏。

业务需求文档（BRD）、功能规约（FS）、测试条件/案例、测试数据和测试中发现的缺陷都是可追溯性矩阵的重要组成部分。下面举例说明了这些组成部分是如何通过可追溯性矩阵集成的，如图17-1所示。

业务需求文档中按照用户细化出来的需求也许并不会完全转化成功能规约中的内容。因此，功能规约和业务需求之间的详细联系是以一对一的形式生成的。这有助于找到文档之间的差异。功能规约的作者将把这些差异关闭掉或者讨论之后推迟到下一个发布版本。最终的功能规约可能会和一开始的版本有所不同，因为无论引入还是推迟一个差异都可能会对应用程序造成连

锁反应。有些时候，这些连锁反应可能没有被恰当地记录下来。这是第一个层次上的可追溯性。

功能规约还将被分解为更小的模块、功能和测试条件来过滤出测试用例，在这些测试用例里给条件输入不同的数据值来验证它们。每个测试条件都是从功能规约中抽取出来的可测试需求。每个测试条件都有一个或多个测试用例和它相关联。每个测试条件都能追溯到它所测试的原始需求。功能规约和测试条件文档之间的这种联系就是第二个层次上的可追溯性。

测试用例是为了验证当前测试的应用程序的一个特定功能而开发的一组测试输入、执行条件和预期结果。每个测试条件的测试用例数量可能是一个或多个。这些测试用例中的每一个都可以追溯到它们所属的测试条件，从而进一步追溯到所测试的原始需求。测试用例和测试条件之间的这种可最终追溯到原始需求的关系就是第三个层次上的可追溯性。

可追溯性的最后一个阶段与测试执行阶段识别出的缺陷相关。将发现的缺陷与测试条件和功能规约相关联，将有助于追溯到需求或者测试条件失败的原因。无论是需求没有描述清楚还是测

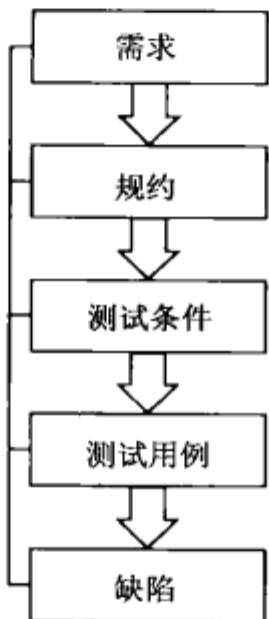


图17-1 可追溯性的树形图

试条件没有从需求中恰当地提炼出来，都可以在将来分配时更正。表17-1举例说明了以上这些交付物是如何通过可追溯性矩阵追踪的。

表17-1 可追溯性矩阵

条目编号	相关文件编号		应用程序/ 模块名称	测试 条件	测试 用例	测试脚本ID	缺陷ID
	业务需求文档编号	功能规约编号					

17.1 用例和可追溯性

用例就是描述参与者（actor）为了实现某一特定目标使用系统的场景。这里的参与者就是为了获得系统某项服务而在系统中扮演角色的一个用户。一般来讲参与者都是人，但是计算机系统也可能是参与者。上面提到的场景就是描述参与者和系统进行交互的一系列步骤。图17-2展示了一幅由所有参与者和所有用例共同组成的用例图。

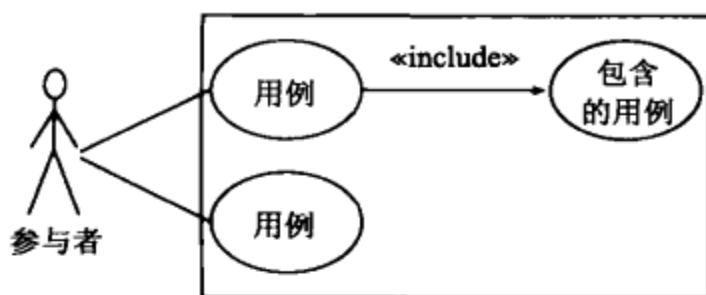


图17-2 用例图

用例：

- 从用户的角度捕获系统的功能需求；
- 在需求收集的过程中使用户积极地参与进来；
- 为识别主要的类和它们之间的关系提供基础；
- 作为开发系统测试用例的基础。

可以在后面的功能规约和前面的测试场景与测试用例文档中追溯到用例。在导出可追溯性的时候应该考虑以下3个问题。

- 用例是否是从系统的最高级别向最低级别展开的。
- 系统的所有功能需求是否都可以对应到用例中。
- 我们是否可以将每个用例都追溯到其所代表的需求。

17.2 小结

在项目的进行过程中总是不断有新的需求加入，也许是因为用户的新增需求，也许是评审过程的结果。这些新增的需求应该恰当地与测试条件和测试用例联系起来。

类似地，在测试应用程序过程中出现的变更请求应当在可追溯性矩阵中得到处理。可追溯性矩阵中的需求任何时候都不应被删除，即使在下个发布版本中删掉该需求也是如此。可追溯性矩阵中的所有需求都应该至少被一个测试用例覆盖。

因此，可追溯性作为一个有效的工具，保证了软件测试的全面性。测试团队测试了所有的需求会增强客户的信心。像Mercury Interactive公司的Test Director软件一样，很多当前流行的测试工具都提供了轻松地创建可追溯性文档的功能。

测试执行/评价（执行/检查）

回忆一下，在螺旋式开发环境中，软件测试可以描述为必须集成到快速应用开发方法中的持续改进过程。应用PDCA模型的Deming持续改进过程原理也被应用于软件测试过程。我们现在探讨螺旋式模型的“执行/检查”部分（见图18-1）。

图18-2列出了与螺旋测试的“执行/检查”部分相关的步骤和任务，描述了每一个步骤和每一项任务，给出了一些有价值的提示和技术。



图18-1 螺旋测试和持续改进

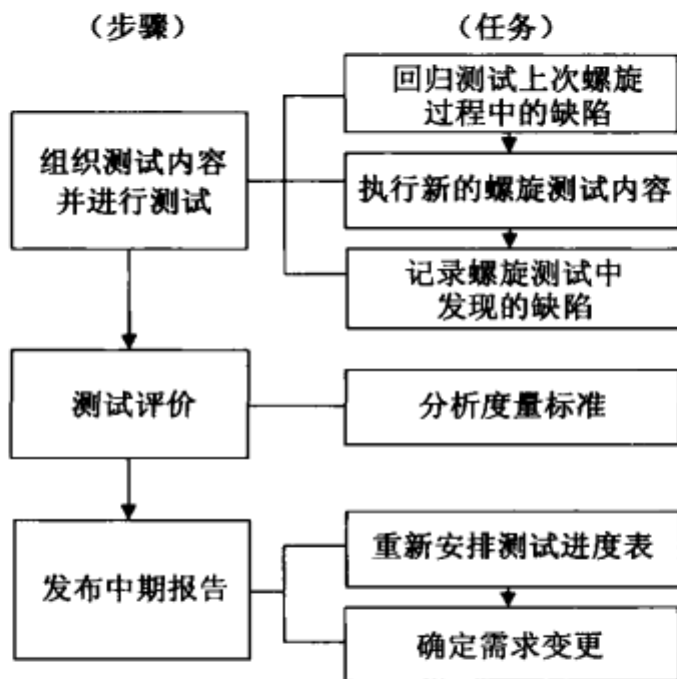


图18-2 测试执行/评价（步骤/任务）

18.1 步骤 1：组织测试内容并进行测试

18.1.1 任务 1：回归测试手工/自动化修复上次螺旋过程中的缺陷

本项任务的目的在于重新测试在上次螺旋测试中发现的缺陷。使用的技术是回归测试。回归

测试是一项用来检测由软件修改或修正引起的欺骗性错误的技术。回归测试的细节参见G.27节。

在软件的整个生命周期中，必须要持续维护一组测试用例，使其一直可用。这组测试用例必须足够完整以便所有软件的功能都被测试，这涉及如何确定测试用例和上一个测试螺旋中发现的缺陷之间的对应关系的问题，再测试矩阵是一个很好的解决方法。

如上所述，再测试矩阵将测试用例与功能（或者程序单元）关联起来。矩阵中的一个检查项都表明，当为了增强和修正而修改某个功能（或程序单元）时，对应的测试用例应该被重新测试。空项代表这个测试用例不需要重新测试。再测试矩阵应该在第一次螺旋开始前先建立好，在后续的螺旋中还需要不断地维护。当功能（或程序单元）在下一次螺旋中被修改了，应该在再测试矩阵中新建相应的测试用例或者检查已经存在的相应测试用例，为下一次螺旋做好准备。随着后面螺旋过程的进行，一些功能（或程序单元）可能最近都没有进行修改，变得很稳定。在测试螺旋之间应当考虑有选择性地从再测试矩阵中删除这些稳定的功能的检查项。

如果通过了回归测试，缺陷报告的状态应该改为“关闭”。

18.1.2 任务 2：执行新的螺旋测试中的手工测试/自动化测试

本项任务的目的是执行在上一次测试螺旋的末尾创建的新测试。在上一次螺旋中，测试团队更新了测试计划、基于图形用户界的功能测试矩阵、测试脚本、图形用户界、阶段性系统测试、验收测试等内容，为当前的测试螺旋做好了准备。在本项任务中，这些测试将被执行。

18.1.3 任务 3：记录螺旋测试中发现的缺陷

在螺旋测试执行的过程中，测试的结果必须在缺陷跟踪数据库中报告。这些缺陷一般与一些单独的测试相关联。然而，正式的测试用例的一些变体经常能够发现其他一些缺陷。本项任务的目的在于生成完整的缺陷记录。如果已经正确地记录下了执行的步骤，那么缺陷也就被记录在缺陷跟踪数据库中了。如果缺陷已经记录下来了，则这个步骤的目的就变成收集整理缺陷信息。

根据测试执行的方法的不同，可以使用不同工具来整理和记录缺陷。如果在纸上记录缺陷，那么整理工作将包括收集并组织这些纸介文件。如果缺陷是用电子文件记录的，那么通过搜索可以很容易找到重复的缺陷。E.27节中给出了一个缺陷报告的例子，用来报告一个特定缺陷的详细情况。

18.2 步骤 2：测试评价

任务：分析度量标准

度量标准用来帮助我们更加有效地做出决策和更好地支持开发过程。本项任务的目标是应用度量标准来控制测试过程。

在前面的任务中，每个螺旋过程要测量的度量标准和度量点已经定义好了。在本项任务中，需要分析所测量的度量，把这些度量的结果进行量化并转化成图形格式。

下面是测试经理在一次螺旋测试结束时需要知道的几个关键信息。

- 测试用例执行状态——多少测试用例被执行了？多少没有执行？多少测试用例发现了缺陷？这是测试人员生产力的一个指标。如果测试用例没有及时执行，就需要给项目分配更多的测试人员。
- 缺陷差距分析——新发现的缺陷数目与被修正的缺陷数目的差距是多少？这是开发人员及时修正缺陷能力的一个指标。如果差距很大，就需要给项目分配更多的开发人员。
- 缺陷严重性情况——缺陷严重性（如严重的、主要的和次要的）的分布是系统质量的一个指标。如果严重缺陷在所有缺陷中占有很大的百分比，就说明系统存在着相当数量的设计和架构上的问题。
- 测试零缺陷跟踪——显示出被发现的缺陷数量的累积变化和周期性变化。例如，累积数量、缺陷总数和某个时间段内的缺陷数量等数据有助于预测从何时开始新发现的缺陷数量越来越少。标志是累积曲线发生“弯折”而且时间段内发现的缺陷数为零。如果累积曲线没有发生弯折的迹象，这就暗示着还有许多的缺陷没有被发现，在后面的螺旋周期中可能会发现更多的缺陷。

第19章中有上述度量标准的图形化示例。

219

18.3 步骤 3: 发布中期报告

E.25节的工程状态报告用来报告关键过程领域中所有测试工程的状态，E.26节的测试缺陷详细情况报告用来报告关键过程领域中所有缺陷的详细状态，E.28节的测试执行跟踪管理是一张Excel电子表格，提供了关于通过/失败的测试用例数量、应用程序中发现的缺陷数量、缺陷状态、完成的百分比、不同种类缺陷的严重性这些方面的综合统计和周期统计。

18.3.1 任务 1: 细化测试进度表

在前面的任务中，已经建立了一个包含测试步骤（可能也包含测试任务）的测试进度表，进度表中还指出了开始时间和结束时间的目标，以及职责。在开发的过程中，需要持续地检查测试进度表。本项任务的目标就是更新测试进度表以反映最新的测试情况。测试经理有责任完成以下任务。

- 将实际进展情况与计划进行对照。
- 对结果进行评价以确定测试状态。
- 在评价结果的基础上采取恰当的行动。

如果测试的进展情况落后于进度，测试经理应当找出造成这种滞后的原因。典型的原因就是对测试工作量的低估。其他原因可能包括发现了超出预计数量的缺陷，导致大量的测试工作量被用于进行已修正缺陷的重复测试。不管是哪种情况，都会需要增加更多的测试人员或者需要延期以弥补滞后造成的影响。

220

18.3.2 任务 2: 确定需求变更

在前面的任务中，功能需求被初始化为要测试的功能，包括分层的功能分解、功能窗口结构、

18

窗口标准以及软件的最低系统需求等。

在两个螺旋之间，新需求可能会被加入到开发的过程中。新需求由以下方面组成：

- 新的GUI接口或组件；
- 新功能；
- 改后的功能；
- 去除的功能；
- 新的系统需求，比如硬件需求；
- 额外的系统需求；
- 额外的验收需求。

221

每个新需求都需要在测试计划、测试设计和测试脚本中得到确认、记录、分析和更新。

第 19 章

准备下一次螺旋测试（改进）

回忆一下，在螺旋式开发环境中，软件测试可以描述为必须集成到快速应用开发方法中的持续改进过程。应用PDCA模型的Deming持续改进过程原理也被应用于软件测试过程。我们现在探讨螺旋式模型的“改进”部分（见图19-1），这部分为下一次螺旋过程做准备。

图19-2列出了与螺旋测试的“改进”部分相关的步骤和任务，描述了每一个步骤和每一项任务，给出了一些有价值的提示和技术。

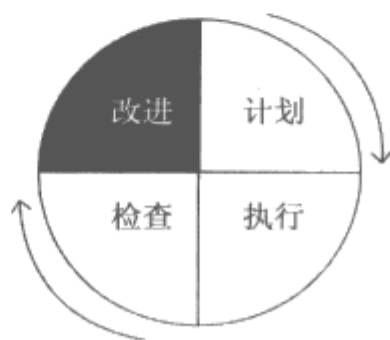


图19-1 螺旋测试和持续改进

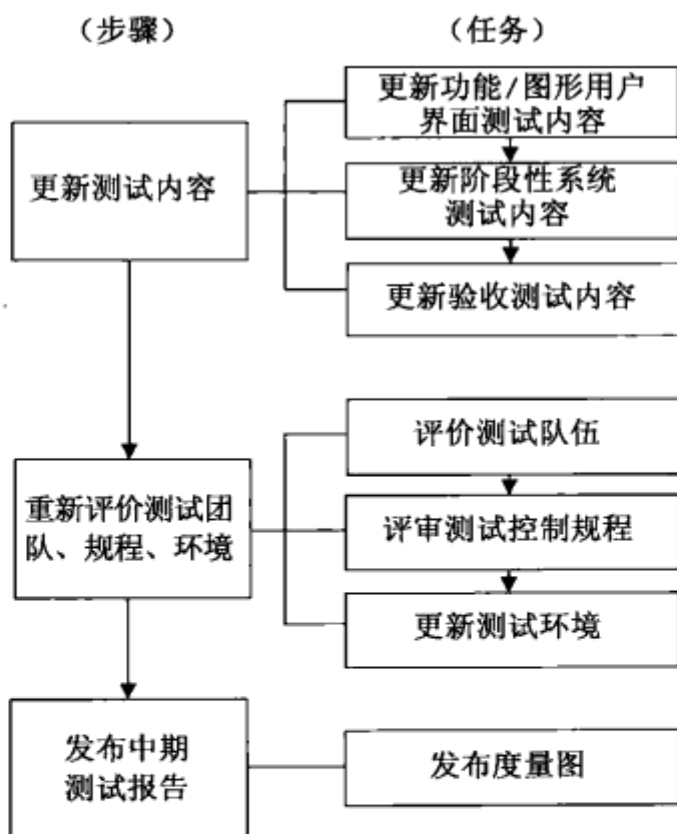


图19-2 准备下一次螺旋测试（步骤/任务）

19.1 步骤 1：细化测试

参照F.21节，影响分析检查表可以用于帮助分析变更对系统产生的影响。

19.1.1 任务 1：更新功能/图形用户界面测试

本项任务的目标是更新测试设计来反映新的功能需求。需要更新用以联系测试和功能的测试变更功能测试矩阵。新的功能应当增加一列，相关的测试用例应当增加一行。测试用例的名称连同编号一起记录到矩阵中（参见E.5节中的功能/测试矩阵）。

接下来，矩阵中任何新加的图形用户界面/功能测试用例都需要被文档化或者脚本化。抽象的测试用例要通过创建的测试数据变成可复用的测试脚本。同样，任何新的图形用户界面的需求也要被加入到图形用户界面测试中（参见E.7节）。

最后，那些可以通过测试工具被自动执行的测试用例也要更新。自动化测试具有以下3点好处：可重复性、杠杆作用、功能可扩展性。可重复性保证了测试可以反复执行多次，每次执行都保持一致。杠杆作用来自于先前执行的那些测试的可重复性，以及可以用工具编排测试的执行顺序，这是自动化测试的独有特征。随着应用程序的不断更新，越来越多的功能被添加进来。使用自动化测试，功能的覆盖通过测试来维护。

19.1.2 任务 2：更新阶段性系统测试

在前面的任务中，阶段性系统测试内容已经定义好了。阶段性系统测试是在每个螺旋循环中都要执行的整个系统测试的样本子集。进行阶段性系统测试的目的在于为将在整个系统测试时可能出现的目前尚未解决的问题提供一个早期的警告。

可选的阶段性系统测试包括功能测试、性能测试、安全性测试、易用性测试、文档测试和规程测试。这些阶段性测试中的一部分需要在每个螺旋过程中都进行正式的测试，而其他的则是整体测试策略的一部分。本项任务的目标是基于新加的需求更新原先创建的阶段性系统测试内容。还应定义新的基线测量。

最后，那些可以通过测试工具自动执行的阶段性系统测试也要更新。

19.1.3 任务 3：更新验收测试

在第15章中，验收测试的初始清单已经定义好了。验收测试是一个由用户执行的可选测试，用来证明应用程序具备满足用户需求的能力。这个测试的出发点是证明功能而不是发现缺陷，也就是说要证明这个系统能够工作。一般来说，验收测试是系统测试的一个子集。然而，用户有时会定义一些“特殊的测试”，比如高强度压力测试或者容量测试，挑战系统的极限，甚至有时会超过系统测试时所测试的。本项任务的目标是基于新加的需求更新原先创建的验收测试内容。

最后，那些通过测试工具可以自动执行的验收测试也要更新。

19.2 步骤 2: 重新评价测试团队、规程和环境

19.2.1 任务 1: 评价测试团队

在螺旋过程中, 测试团队的表现需要以测试质量和生产力为标准来进行评价。测试团队的主管应直接指导一个或多个测试人员以保证测试项目的各个方面都被正确执行。他要确认每个测试用例都按照测试计划中的描述正确执行了, 每个缺陷都准确地被报告并重新测试了, 并且自动化测试都成功地执行了。特定测试资源分配的根据是功能的覆盖范围和开发的时间框架。如果测试工作完成的不尽如人意, 那么团队主管需要和一些团队成员进行交流/或者申请更多的测试人员。另一方面, 如果测试进展到了收尾阶段, 测试经理应该开始考虑将测试人员调整到其他的项目中去。

225

19.2.2 任务 2: 评审测试控制规程

在第14章中, 在第一次螺旋过程开始前已经建立好了测试的控制规程。本项任务的目标就是评审这些流程并做出恰当的修改。预先定义好的规程包括以下几个部分。

- 缺陷记录/跟踪规程。
- 变更请求规程。
- 版本控制规程。
- 配置构建规程。
- 项目问题解决规程。
- 报告规程。

缺陷记录/跟踪规程的目的是记录并更正缺陷, 并且记录该应用的度量信息。随着项目的进展, 这些规程可能都需要做适当的调整。例如, 缺陷跟踪的格式中添加了新状态码或者新字段, 扩展了缺陷分布清单, 或者新增了验证检查等。

变更请求流程的目的是将新的变更请求通知给开发团队和测试团队。随着项目的进展, 变更请求规程可能都需要做适当的调整。例如, 新的变更控制评审会议、对变更请求规程应该如何实现有新想法的新负责人、新变更请求数据库以及新的软件配置管理工具等。

版本控制规程的目的是通过一种标号方案对所有软件组件进行唯一标识, 并且考虑连续的版本。随着项目的进展, 版本控制流程可能都需要做适当的调整。例如, 出现新的软件配置管理工具, 其中包含新的版本区分方案或者新的标号标准等。

配置构建规程的目的是提供一种装配软件系统的有效方法, 能够将软件源代码组件变成可执行的组件。随着项目的进展, 配置构建流程可能都需要做适当的调整。例如, 添加新的第四代编程语言、新的软件配置管理工具或者Delta构建方法等。

项目问题解决规程的目的是记录并处理在软件测试过程中发现的问题。随着项目的进展, 项目问题解决规程可能都需要做适当的调整。例如, 需要Lotus Notes的新项目经理、新成立的问题评审委员会、更新问题的优先权分类方案或者新的问题提交过程等。

226

报告规程的目的是促进交流过程和报告的编写。随着项目的进展, 报告规程可能都需要做适当的调整。例如, 要求每周一份测试状态报告的新项目经理、新的中期测试报告结果或者扩展的报告分布状态。

19.2.3 任务3：更新测试环境

在第15章中，测试环境已经定义好了。测试环境提供了测试活动所必需的物理框架。在本项任务中，需要评审和更新测试环境的需求。（详见F.22节，环境准备检查表可以用来在开始测试执行前验证测试环境的就绪情况。）

测试环境的主要组成部分包括物理测试设备、测试技术和测试工具。物理测试设备部分主要指物理设备的安装。测试技术部分包括使用的硬件平台、物理网络及其组件、操作系统软件和一些其他的软件，比如一些工具软件等。测试工具部分包括一些特殊的测试软件，比如自动化测试工具、测试库以及一些相关支持软件。测试环境的变更有如下几种情况。

- 扩充的测试实验室。
- 新的测试工具需求。
- 附加的测试硬件需求。
- 附加的网络设备。
- 附加的测试资料库空间需求。
- 新的Lotus Notes登录许可。
- 附加的测试支持软件。

19.3 步骤3：发布中期测试报告

任务：发布度量图

每次螺旋都应该生成一份中期报告来描述当前测试的状态。这些测试报告将发送给测试团队、测试经理和开发经理，并且帮助他们为下一次螺旋过程做出调整。每次螺旋测试过程结束时都推荐生成以下4个基本图形报告。

1. 测试用例执行状态

图19-3显示了测试的状态并预计出何时测试和开发团队能够做好准备继续工作。测试用例运行时发现的错误还没有被更正。

227

如果没有运行的测试用例在所有测试用例中占有相当多的数量，那么测试团队需要提高生产力或者增加资源。如果测试用例发现的问题还有很多没有更正，那么开发团队同样需要提高一下生产效率。

2. 缺陷差距分析

图19-4显示了已发现的缺陷数与已更正的缺陷数之间的差距。如果存在很大的差距就说明开发团队需要增加工作量和资源以加快缺陷修正的速度。

3. 缺陷严重性情况

图19-5显示了3种严重级别（严重的、主要的和次要的）的缺陷的分布情况。如果严重层次上的缺陷占很高的比例，就说明应用程序的设计或架构上可能存在问题。

4. 测试零缺陷跟踪

图19-6显示了已发现缺陷的速度。例如，累积数量、缺陷总数和某个时间段内的缺陷数量等

数据有助于预测从何时开始新发现的缺陷数量越来越少。标志是累积曲线发生“弯折”而且时间段内发现的缺陷数为零。

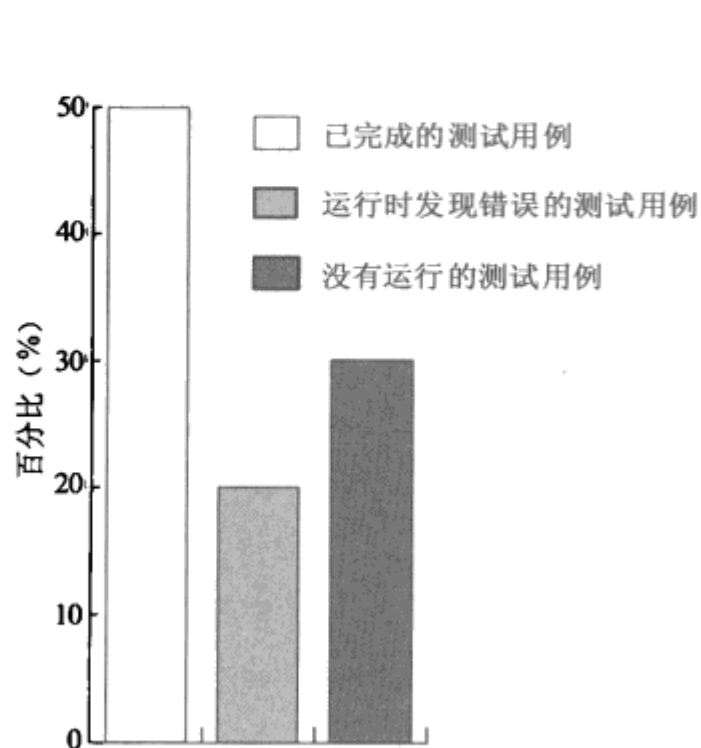


图19-3 测试执行状态

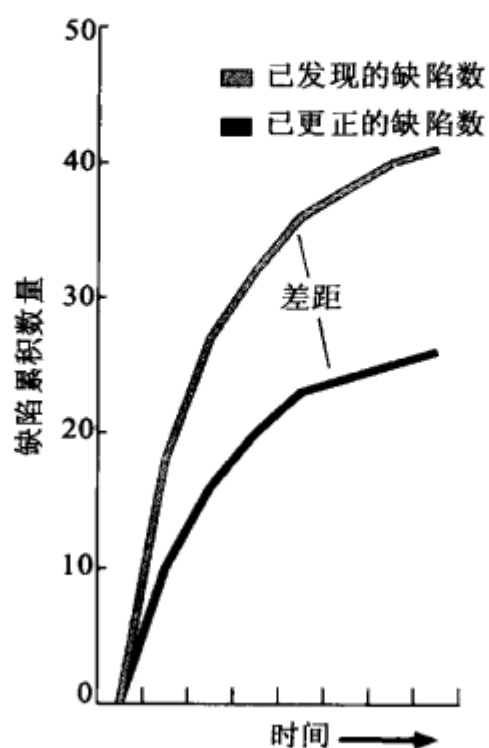


图19-4 缺陷差距分析

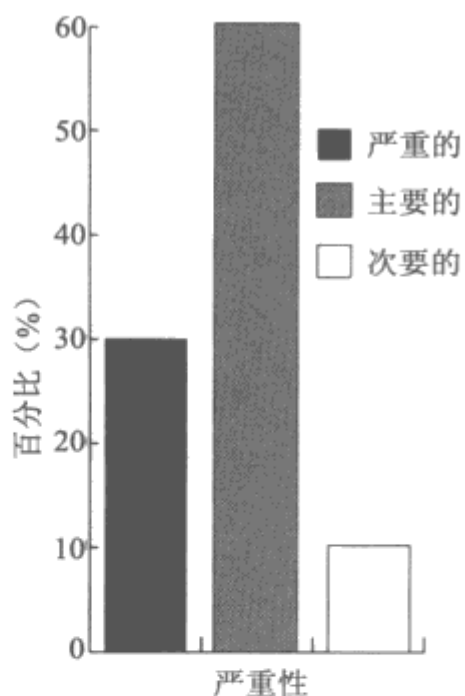


图19-5 缺陷严重性情况

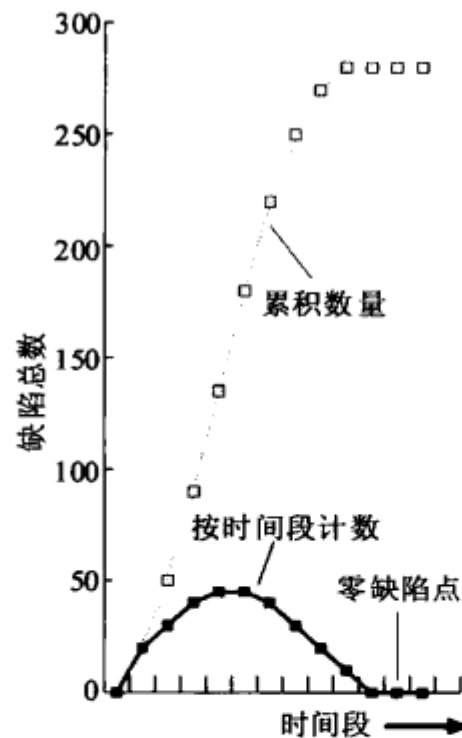


图19-6 测试零缺陷跟踪

系统测试对整个应用程序的功能和性能加以评估，由很多种测试组成，包括性能测试、易用性测试、压力测试、文档测试、安全性测试、容量测试、可恢复性测试等。图20-1描述了如何对阶段性系统测试加以扩展，图中讨论了如何准备系统测试，如何设计系统测试并将其脚本化，如何执行系统测试，以及如何报告在测试过程中发现的异常。

20.1 步骤 1：完成系统测试计划

20.1.1 任务 1：确定系统测试类型

在前一个任务中的每个螺旋内，我们都选择并执行了一组阶段性系统测试。当前任务的目的是最终决定在系统测试中将进行的测试类型。

如果你还记得的话，系统的初始目标是在项目访谈时定义的，而系统测试是由一个或多个基于该目标的测试组成的。当前任务的目的是选择将要进行的系统测试而不是实现这些测试。我们的系统测试的初始清单包括以下系统测试类型。

- 性能测试。
- 安全性测试。
- 容量测试。
- 压力测试。
- 兼容性测试。
- 转换测试。
- 易用性测试。
- 文档测试。
- 备份测试。
- 可恢复性测试。
- 安装测试。

在本项任务中也应当定义不同类型的系统测试的执行顺序。例如，像性能测试、压力测试、容量测试这样的相关测试可以放在一组并在系统测试的早期进行，安全性测试、备份测试和可恢复性测试逻辑上也可分为一组，等等。

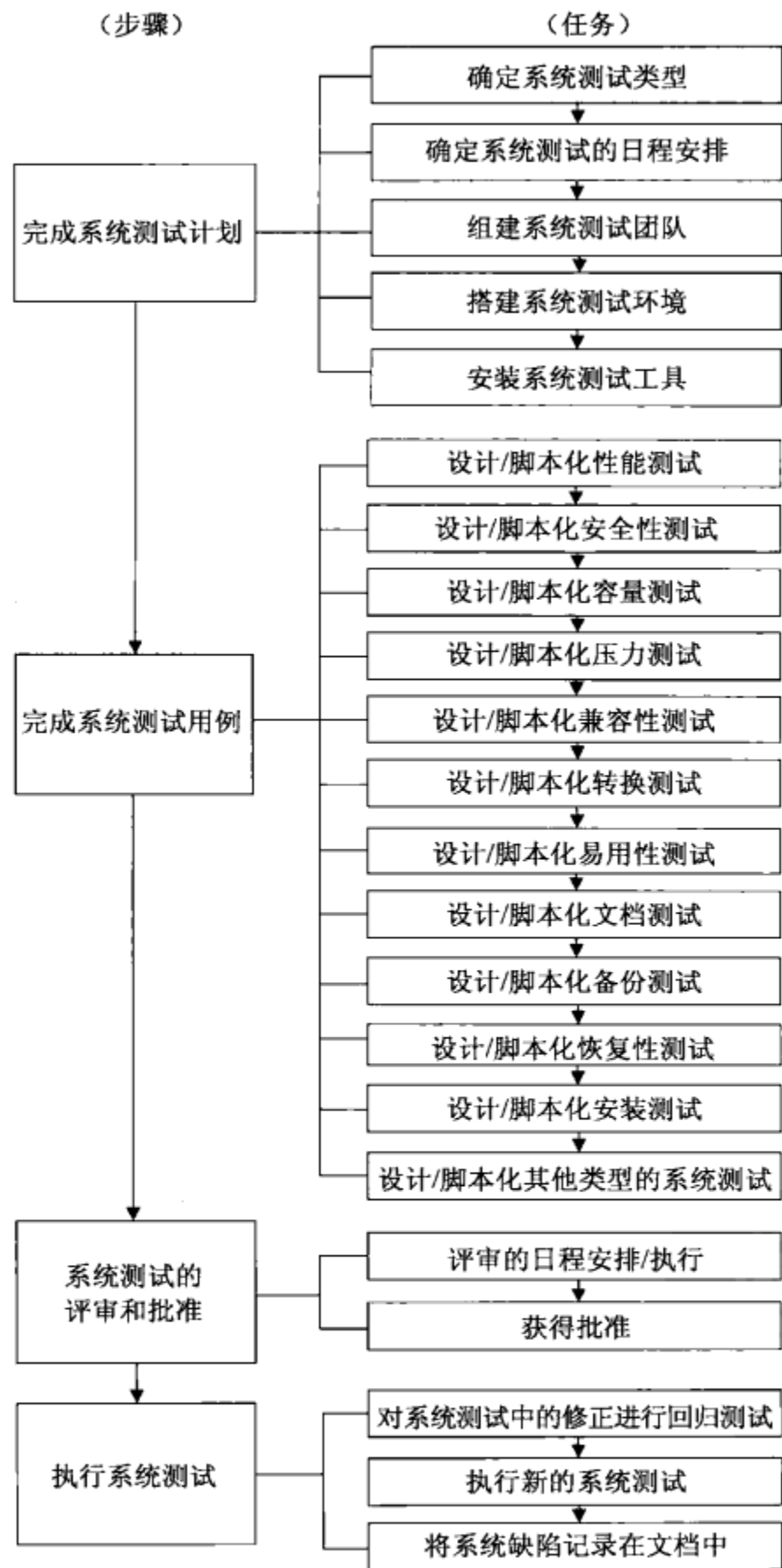


图20-1 进行系统测试（步骤/任务）

最后，需要确定可以使用测试工具自动进行的系统测试。自动化测试具有以下3点好处：可重复性、杠杆作用、功能可扩展性。可重复性保证了测试可以反复执行多次，每次执行都保持一致。杠杆作用来自于先前执行的那些测试的可重复性，以及可以用工具编排测试的执行顺序，这

是自动化测试的独有特征。随着应用程序的不断更新,越来越多的功能被添加进来。使用自动化测试,功能的覆盖通过测试库来维护。

20.1.2 任务2: 确定系统测试的日程安排

在本项任务中,要最终决定系统测试的日程安排,包括测试步骤(甚至还可能包括任务)、目标开始日期和目标结束日期及人员职责。本项任务还应该描述系统测试将如何被评审、追踪和批准。表20-1中所示是一个系统测试进度表的例子。

表20-1 最终的系统测试进度表

测试步骤	开始日期	结束日期	负责人
总体启动			
组建系统测试团队	12/1/04	12/7/04	Smith, 测试经理
搭建系统测试环境	12/1/04	12/7/04	Smith, 测试经理
安装系统测试工具	12/1/04	12/10/04	Jones, 测试人员
性能测试			
设计/脚本化测试	12/11/04	12/15/04	Jones, 测试人员
测试评审	12/16/04	12/16/04	Smith, 测试经理
执行测试	12/17/04	12/22/04	Jones, 测试人员
重新测试系统缺陷	12/23/04	12/25/04	Jones, 测试人员
压力测试			
设计/脚本化测试	12/26/04	12/30/04	Jones, 测试人员
测试评审	12/31/04	12/31/04	Smith, 测试经理
执行测试	1/1/04	1/6/04	Jones, 测试人员
重新测试系统缺陷	1/7/04	1/9/04	Jones, 测试人员
容量测试			
设计/脚本化测试	1/10/04	1/14/04	Jones, 测试人员
测试评审	1/15/04	1/15/04	Smith, 测试经理
执行测试	1/16/04	1/21/04	Jones, 测试人员
重新测试系统缺陷	1/22/04	1/24/04	Jones, 测试人员
安全性测试			
设计/脚本化测试	1/25/04	1/29/04	Jones, 测试人员
测试评审	1/30/04	1/31/04	Smith, 测试经理
执行测试	2/1/04	2/6/04	Jones, 测试人员
重新测试系统缺陷	2/7/04	2/9/04	Jones, 测试人员
备份测试			
设计/脚本化测试	2/10/04	2/14/04	Jones, 测试人员
测试评审	2/15/04	2/15/04	Smith, 测试经理
执行测试	2/16/04	1/21/04	Jones, 测试人员
重新测试系统缺陷	2/22/04	2/24/04	Jones, 测试人员

(续)

测试步骤	开始日期	结束日期	负责人
可恢复性测试			
设计/脚本化测试	2/25/04	2/29/04	Jones, 测试人员
测试评审	2/30/04	2/31/04	Smith, 测试经理
执行测试	3/1/04	3/6/04	Jones, 测试人员
重新测试系统缺陷	3/7/04	3/9/04	Jones, 测试人员
兼容性测试			
设计/脚本化测试	3/10/04	3/14/04	Jones, 测试人员
测试评审	3/15/04	3/15/04	Smith, 测试经理
执行测试	3/16/04	3/21/04	Jones, 测试人员
重新测试系统缺陷	3/22/04	3/24/04	Jones, 测试人员
转换测试			
设计/脚本化测试	4/10/04	4/14/04	Jones, 测试人员
测试评审	4/15/04	4/15/04	Smith, 测试经理
执行测试	4/16/04	4/21/04	Jones, 测试人员
重新测试系统缺陷	4/22/04	4/24/04	Jones, 测试人员
易用性测试			
设计/脚本化测试	5/10/04	5/14/04	Jones, 测试人员
测试评审	5/15/04	5/15/04	Smith, 测试经理
执行测试	5/16/04	5/21/04	Jones, 测试人员
重新测试系统缺陷	5/22/04	5/24/04	Jones, 测试人员
文档测试			
设计/脚本化测试	6/10/04	6/14/04	Jones, 测试人员
测试评审	6/15/04	6/15/04	Smith, 测试经理
执行测试	6/16/04	6/21/04	Jones, 测试人员
重新测试系统缺陷	6/22/04	6/24/04	Jones, 测试人员
安装测试			
设计/脚本化测试	7/10/04	7/14/04	Jones, 测试人员
测试评审	7/15/04	7/15/04	Smith, 测试经理
执行测试	7/16/04	7/21/04	Jones, 测试人员
重新测试系统缺陷	7/22/04	7/24/04	Jones, 测试人员

20.1.3 任务 3: 组建系统测试团队

在不同的测试类型中，系统测试团队的组织结构也是不尽相同的。系统测试团队负责设计和执行系统测试、评价测试结果、使用缺陷跟踪系统向开发团队报告发现的所有缺陷，当开发修正了缺陷之后，测试团队需要对这些缺陷重新进行测试以确认修正是正确无误的。

系统测试团队由一个测试经理领导，其职责包括：

- 组建系统测试团队；

- 搭建测试环境;
- 组织建立测试的政策、规程和标准;
- 确保测试准备就绪;
- 运行测试计划并对项目进行控制;
- 对各项测试成本进行追踪;
- 确保测试文档的准确性和及时性;
- 管理团队成员。

20.1.4 任务 4：搭建系统测试环境

在本项任务中，要最终决定系统测试的环境。搭建测试环境的目的是为测试活动提供一个物理框架。测试环境需要在实现前建成并通过评审。

测试环境的主要组成部分包括物理测试设备、测试技术和测试工具。物理测试设备部分主要是指物理设备的安装。测试技术部分包括硬件平台、物理网络及其组件、操作系统软件以及其他的一些软件。测试工具部分包括各种专用的测试软件，比如自动化测试工具、测试库和一些支持软件等。

除此之外，还需要安装测试设备和建立工作间，可能包括从一个独立工作间的配置到一个正式的测试实验室的创建等工作。但是无论如何，让测试人员能够一起工作并且距离开发团队不能太远非常重要，有利于彼此的沟通，对培养项目组的共同目标感也很有好处。同时系统测试工具也需要安装。

硬件和软件技术也需要准备，这包括安装测试所需的硬/软件，以及协调厂商、用户和信息技术人员。测试硬件和协调硬件厂商也是必需的。通信网络也需要安装和测试。

20.1.5 任务 5：安装系统测试工具

在本项任务中要安装系统测试工具并检验以确保其准备就绪。通过运行所需的测试用例和脚本确保测试工具已经可用于实际的验收测试。下面是需要考虑的另外一些关于工具就绪方面的因素。

- 培训测试团队如何使用工具。
- 工具与操作环境的兼容性。
- 足够的硬盘空间以满足工具使用的需要。
- 最大限度发挥的工具潜能。
- 厂商的工具帮助热线。
- 修改测试规程使之适用于测试工具。
- 安装工具的最新更新包。
- 确保厂商遵守契约条款。

20.2 步骤 2：完成系统测试用例

在这个步骤中要设计系统测试用例并将其脚本化。概念上的系统测试用例被转换成为可复用

的测试脚本, 同时创建测试数据。

为了协助进行测试用例脚本的开发, 可以使用E.7节中的基于图形用户界面的功能测试矩阵模板来记录系统级测试用例, 注意把标题中的“功能”两个字替换成系统测试的名字。

20.2.1 任务 1: 设计/脚本化性能测试

性能测试的目标是根据预先定义的目标对系统进行度量, 需要将需要的性能等级与实际性能等级进行对比, 并将差异记录在文档中。

性能测试是黑盒测试与白盒测试的混合。从黑盒测试的观点来看, 性能分析不需要知道系统的内在运行情况。实际的工作负载或基准点可用于比较系统一个版本与另一个版本之间性能方面的差异。从白盒测试观点来看, 性能分析需要了解系统的内在运行机制, 并定义需要进行检测的系统资源, 比如指令、模块和任务等。

下面是一些重要的性能信息。

- CPU的利用率。
- IO的利用率。
- 每条指令的I/O数。
- 信道的利用率。
- 内存的利用率。
- 外部存储空间的利用率。
- 每个模块执行时间所占的百分比。
- 一个模块等待I/O操作完成所用时间的百分比。
- 模块占用内存时间的百分比。
- 在时间轴上对指令的追踪路径。
- 控制权从一个模块传递到另一个模块的次数。
- 每一组指令遇到的等待次数。
- 每一组指令的页面换入/换出次数。
- 系统响应时间, 例如最后一个关键点到第一个关键点的时间。
- 系统吞吐量, 即每个单位时间内处理的事务数。
- 所有主要功能的单位性能计时。

进行基线性能测量时应当首先确保所有的主要功能处于一种非竞争操作模式下, 例如, 对某个功能进行的单位测量就是单独运行这个任务。这种测量用一个简单的计时器即可, 如同在每个螺旋过程的早期所做的那样。接下来的是在系统的竞争模式下进行性能测量, 在这种模式下, 多个任务同时执行并且排队等待共用资源, 比如CPU、内存、硬盘、信道、网络等。通过监视系统以确定效率低下的潜在区间的方式, 可以对竞争系统执行时间和资源利用率进行性能测量。

收集系统执行时间和资源利用率有两种方法: 其一, 当系统正在其典型环境下运行时, 使用外部探测、性能监视器或计时器等进行取样; 其二, 将探测代码插入系统代码中, 例如, 调用一个收集性能信息的性能监控程序。下文中对这两种方法都有讨论, 再接下来是对测试驱动的讨论, 测试驱动技术是用来支持产生性能测试数据的。

1. 监视法

240

这种方法通过在周期性时间间隔测定系统状态来对其进行监视,并由测试工具中的或操作系统中的耗用时间指示设施加以控制。在每一个时间间隔进行的取样表明了该间隔中性能标准的状态。时间间隔越短,取样精确率越高。

通过监视取得的统计数据在执行过程中收集并汇总。

2. 探测法

这种方法在系统程序的不同位置插入探测代码或者程序指令。例如,为了测定执行一个语句序列所必需的CPU时间,调用一个记录瞬时CPU时钟的数据收集例程,随后再进行探测,第二次调用该数据收集例程,将两次记录的时间相减,就得到使用的净CPU时间。可以按语句、模块或语句类型显示执行时间分解来生成报表。

这些方法的价值在于它们可以用作性能需求的验证工具。然而,正式定义的性能需求必须有所规定,并且应该对系统进行设计,以使得性能需求可以被追踪到特定的系统模块中去。

3. 测试驱动

在许多情况下,需要用测试驱动程序和测试执行器来进行系统性能的测量。测试驱动程序用来提供执行一个系统所需的“基础设备”,比如系统的输入等。系统的数据输入文件和表示测试情景的数据值一起被载入到系统中,然后对比执行结果和预期结果。数据均使用统一的表现格式输入到系统中。

性能测试用例需要使用附录中的一个或多个测试模板进行定义,并且需要建立测试脚本。然而,在进行任何性能测试之前,性能分析人员必需确保目标系统是相对无缺陷的,否则需要花费大量的时间记录和修复缺陷,而无法进行性能分析。

下面是性能研究的5个推荐步骤。

(1) 将性能目标记录在文档中;例如,必须验证哪些可量度的性能标准。

(2) 定义测试驱动程序或者能够驱动系统的输入源。

(3) 定义将使用的性能测试方法或工具。

(4) 定义如何进行性能研究;例如,基线是什么,差异是什么,如何能证明其可重复性,以及如何确定性能测试已经完成,等等。

241

(5) 定义报告过程,如技术和工具等。

20.2.2 任务2:设计/脚本化安全性测试

安全性测试的目标是对应用程序安全性的表现和相应的安全性机制加以评价,以保证数据的完整性和机密性。安全性测试应该仔细设计以证明资源的保护机制是合理的。

安全性设计策略

设计出安全性测试用例的策略是把重点放在以下4个安全性要素上:资产、威胁、风险和控制。用这个方式,可以从矩阵和检查表得到一些安全性测试用例的设计建议。

资产是有形的和无形的资源实体。其评估方法是列出需要保护的内容。检查资产的属性也是十分有用的,例如数量、价值、用途和特性等。两种有用的分析技术是资产价值分析和资产开发分析。资产价值分析确定资产对用户和潜在攻击者的价值的差异。资产开发分析检查将资产用于

非法获利的各种方法。

威胁是潜在的可能造成损失或危害的事件。其评估方法是列出潜在威胁的来源。分辨偶然威胁、故意威胁、自然威胁和威胁发生的频率是非常重要的。

风险包括可能的损失或危害。其评价方法是列出如果一个威胁真的出现会对资产发生什么影响。风险包括违规泄漏、错误决定和欺诈。风险分析的重点是标识出风险最高的区域。

安全功能或安全控制是为了保护资产免遭损失或危害。其评价方法是列出安全功能和任务并通过特定的系统功能或规程集中控制。安全功能评定保护资产免遭人为破坏和偶然误用破坏。一些程式化的安全性问题包括以下内容。

- 控制特性是否正常运转？
- 无效的和不太可能正确的参数是否被检测出来并适当处理了？
- 无效的或不在执行序列之内的命令是否被检测出来并适当处理了？
- 错误和文件访问是否都适当地记录下来？
- 改变安全表格的规程是否能起到作用？
- 是否可能不使用密码登录？
- 有效的密码是否被接受？无效的密码是否被拒绝？
- 系统是否对多个无效的密码做出适当的反应？
- 系统是否能适当地初始化认证功能？
- 对于远程访问是否具有安全特性？

评价安全机制的性能同评价其功能一样重要。一些关于安全性能的问题包括以下内容。

- 可用性——应用或者控制可用于完成关键的安全功能的时间百分比是多少？安全控制通常需要比系统其他部分拥有更高的可用性。
- 可靠性——系统抵御致命故障或自然灾害的能力如何？这包括对故障时的紧急操作、事后备份和还原到正常状态的恢复动作等的支持。
- 精确性——安全控制有多精确？精确性包含错误的数量、频率和重要性。
- 响应时间——响应时间是否在可接受范围内？缓慢的响应时间可能使用户绕开安全控制。响应时间对控制管理也是至关重要的，例如对安全表格的动态修改。
- 吞吐量——安全控制是否支持所需的使用容量？容量包括用户和服务请求的峰值和平均负载。

压力测试是一种有用的性能测试，它涉及很大数量的用户和请求造成的操作压力状况。压力测试用于某些资源的极限逼近测试，如缓冲区、队列、表和端口等。这种测试形式对评价对拒绝服务威胁的保护是很有用的。

20.2.3 任务 3: 设计/脚本化容量测试

容量测试的目标是使系统承受大容量数据以检查系统是否能够处理这么大的容量。这种测试经常与压力测试相混淆。压力测试是将系统置于重负载或快速访问的压力下，例如测试系统在一个短周期内的吞吐量。而容量测试是面向数据的，它的目的是显示系统能够处理其目标中指定的数据容量。

下面是一些容量测试的例子。

- 当处理数据密集型事务时比较相关的数据。
- 使用编译器编译一个异常大的源程序。
- 使用链接器链接一个包括数千个模块的程序。
- 给电路模拟器一个包括数千个元件的电路。
- 将操作系统的任务队列填充到最大容量。
- 创造足够多的数据使系统超过文件容量。
- 使用格式测试系统处理大量的文本格式。
- 使网络充斥着巨大的电子邮件消息和文件等。

20.2.4 任务 4：设计/脚本化压力测试

压力测试的目标是检查系统在资源超载的情况下的行为，其中特别重要的一点是其对系统处理时间的影响。压力测试是边界测试。例如，使用最大数目的激活终端进行测试，然后在各种不同的限制组合下加入比需求中指定的要多的终端。下面是一些在压力测试下加以很重负载的资源。

- 缓冲区。
- 控制器。
- 显示终端。
- 中断处理器。
- 内存。
- 网络。
- 打印机。
- 调度器。
- 存储设备。
- 事务队列。
- 事务调度器。
- 系统用户。

压力测试研究系统对短时间内爆发的最大量活动的反应，并试图找出系统内的缺陷。压力测试经常与容量测试相混淆，实际上容量测试的目标是系统处理很大容量数据的能力。

压力测试应该在开发早期进行，因为它经常可以发现一些主要的设计缺陷，这些缺陷可能对多个区域具有影响。如果没有及早进行压力测试，那么在开发早期更为明显的一些细微的缺陷可能很难发现。

下面是压力测试的建议步骤。

- (1) 进行简单的多任务测试。
- (2) 在简单的压力缺陷得到纠正后，向系统施加压力到其临界点。
- (3) 在每个螺旋测试周期中反复进行压力测试。

下面是一些压力测试的例子。

- 对固定输入速度（如120字每分钟）的字处理响应时间。

- 在非常短的时间段内引入非常大量的数据。
- 改变交互的实时过程控制的负载。
- 同时引入很大数量的事务。
- 在一分钟内有数以千计的用户登录到因特网。

20.2.5 任务 5: 设计/脚本化兼容性测试

兼容性测试（有时候也被称为共栖性测试）的目标是测试一个应用程序与其他应用程序或系统之间的兼容性。在系统产品化之前这个测试经常被忽视，在这种测试中，缺陷经常非常细微，很难被发现。一个例子是系统在测试实验室的控制环境下运行完全正常，而当它与其他应用程序共存时却无法运行。兼容性的另一个例子是两个系统同时共享同一数据或数据文件，或者同时占用同一块内存。系统也许可以满足系统需求，但是无法在共享环境下运行，并且可能与其他系统相互影响。

下面是一种兼容性（共栖性）测试策略。

(1) 更新兼容性目标以记录应用程序到底是如何开发的，以及它将要实际运行的环境。一旦共栖系统或配置资源有所变动，则相应修改这一目标。

(2) 更新兼容性测试用例以保证其全面性。保证测试用例可以全面涉及所有可能影响本系统的其他系统，并且保证能够最大程度地覆盖本系统对其他软件系统可能造成的影响。

(3) 运行兼容性测试并仔细检查其结果，以保证可以得到预期的结果。使用基线方法，即需要记录目标系统加入共享环境之前的系统操作特性。基线需要非常准确，并且不仅包括其功能，还要包括其操作性能，以保证在共栖的设置中基线水平不会降低。

(4) 把兼容性测试的结果记录在文档中，并注明任何目标系统或共栖系统中的偏离。

(5) 在缺陷都已经解决后进行兼容性测试的回归测试，并将测试结果记录在再测试矩阵中。

20.2.6 任务 6: 设计/脚本化转换测试

转换测试的目标是对现有数据的转换和载入一个新的数据库加以验证。最常见的转换问题出在同一个系统的两个版本之间。新版本可以使用新的数据格式，但是必须包括从旧系统中继承的数据。需要留出充足的时间来仔细考虑可能产生的所有转换问题。

在设计转换测试的时候需要考虑的一些关键要素如下所述。

- 可审计性——需要有一个进行执行之前与执行之后比较的计划，并且需要对转换数据加以分析，以保证数据得以成功转换。保证可审计性的技术包括文件报告、比较程序和回归测试。回归测试检验确保转换过的数据没有改变业务需求或者造成系统运行异常。
- 数据库验证——在转换之前，需要检查新的数据库以保证它是设计适当、能够满足业务需要的，并且支持中心和数据库管理员都已经过训练能够为其提供支持。
- 数据清理——在数据被转换到新系统中之前，需要对旧数据加以检查，以保证不准确的和自相矛盾的部分得以清除。
- 恢复计划——在试图进行任何转换之前，回滚程序必须就位，将系统恢复到它以前的状态并能撤销转换操作。

- 同步——必须验证转换过程没有干扰到正常的操作。类似于客户数据这样的敏感数据可以在转换中动态地加以改变。达到这个目的的一种方法是在没有进行操作的时间段内进行转换工作。

20.2.7 任务 7：设计/脚本化易用性测试

易用性测试的目标是测定用户能够多好地使用和理解应用程序。这包括保证用户能够舒适地与系统进行交互的系统功能、出版物、帮助文本规程。易用性测试应当在开发过程中尽可能早地进行，并且应当被设计在系统中。到了开发后期才进行易用性测试是不可能的，因为这时易用性已经确定了，而且要纠正严重的易用性问题经常需要对系统进行一个重大的架构调整，这从经济方面来说也是不可行的。

下面是测试人员应当注意的一些易用性问题。

- 过度复杂的功能或使用说明。
- 较难的安装过程。
- 不够完善的错误消息，例如，“语法错误”。
- 语法难于理解和使用。
- 非标准化的图形用户界面。
- 强迫用户记住过多的信息。
- 较难的登录流程。
- 帮助文本是对上下文不敏感的，或者不够详细。
- 与其他系统的连接不够完善。
- 不清晰的默认值。
- 界面过于简单或者过于复杂。
- 语法、格式和定义前后不一致。
- 没有向用户提供对所有输入的明确的确认。

20.2.8 任务 8：设计/脚本化文档测试

246

文档测试的目标是核实用户文档的准确性并保证手册中的规程是正确的。文档测试具有几个优势，包括提高系统的易用性、可靠性、可维护性和可安装性等。在这些情况下，对文档进行测试将有助于发现系统中的不足/或者使系统更加易于使用。

文档测试也减少了客户支持的费用，比如当客户能够使用文档解决自己的问题时，他们就不必非得给支持人员打电话了。

测试人员需要验证文档的技术准确性以确保文档是符合系统应用的，并确保对系统的描述是准确的。这需要测试人员将自己置于用户的角度，并完全按照文档中描述的行为加以操作。

下面是给文档测试人员的一些提示和建议。

- 将文档作为许多测试用例的来源。
- 完全按照文档中的描述使用系统。
- 对每个提示和建议加以测试。

- 将缺陷加入缺陷跟踪数据库中。
- 对每个在线帮助超文本链接加以测试。
- 对事实的每一个描述加以测试，并且不要想当然地看待任何事情。
- 要像一个技术编辑，而不是一个被动的评审者。
- 首先对整个文档进行全面的评审，然后再评审细节。
- 检查所有的错误消息。
- 对文档中给出的所有例子加以测试。
- 保证所有的索引项都有对应的文档正文。
- 保证文档覆盖了所有关键的用户功能。
- 保证阅读的文体不是过于技术化。
- 注意寻找比其他地方更薄弱和需要更多解释之处。

20.2.9 任务 9: 设计/脚本化备份测试

备份测试的目标是验证系统在遭遇软件故障或硬件故障时备份其数据的能力。这个测试是对可恢复性测试的补充，应当是可恢复性测试计划的一部分。

下面是备份测试需要考虑的一些事项。

- 备份文件并且对备份文件与原文件进行比较。
- 对文件和数据进行归档。
- 完整的系统备份规程。
- 对检查点的备份。
- 备份性能系统降级。
- 备份对手动过程的影响。
- 对系统备份的触发条件的检测。
- 在备份过程中的安全性规程。
- 在备份规程期间对事务日志的维护。

247

20.2.10 任务 10: 设计/脚本化恢复性测试

恢复性测试的目标是验证系统从软件故障或硬件故障中恢复的能力。这个测试验证了系统运行过程中对中断处理和回溯到某个特定时间点的应急特性。下面是设计可恢复性测试的关键问题。

- 灾难、系统故障和它们各自的损害的可能性是否已经确定？消防演习式的头脑风暴会议可以作为定义灾难情景的有效方法。
- 预防和恢复规程是否足以应付故障？计划规程应当由有关专家和系统用户进行技术评审。
- 在真正需要的时候恢复规程是否能够正常地运行？应当对实际系统创建模拟灾难以验证恢复流程是否能够正常运行。这涉及系统用户、支持组织、厂商等。

下面是可恢复性测试的一些例子。

- 完整地恢复一份在日常维护或错误恢复过程中备份的文件。
- 把备份文件部分地恢复到最近的检查点。
- 执行恢复程序。
- 选定文件或数据的归档检索。
- 当电源出问题后进行恢复。
- 验证手动恢复规程。
- 通过切换到并行系统进行恢复。
- 恢复性能系统降级。
- 恢复过程中的安全规程。
- 恢复事务日志的能力。

20.2.11 任务 11：设计/脚本化安装测试

安装测试的目标是验证成功地安装系统的能力。客户需要将产品安装到自己的系统中。安装通常是开发者要做的最后一步，并且经常在开发过程中最少关注的。然而，安装却是客户使用新系统的第一步。因此，清晰、简洁的安装流程是系统文件最重要的部分之一。

248

测试中也要包括重新安装的规程，使反向的安装过程能够进行，并使以前的环境条件生效。并且，安装规程的文档需要给出用户如何调节系统安装选项以及如何从一个以前的版本进行升级。

下面是测试人员需要考虑的一些关键问题。

- 谁为用户安装人员？例如，假设安装者的技术能力是什么水平？
- 安装规程用详细简洁的安装步骤在文档中说明了吗？
- 安装规程应该在什么样的环境（如平台、软件、硬件、网络或版本等）下运行？
- 安装会改变用户现有的环境设置（如config.sys文件等）吗？
- 安装人员如何确认系统已经正确地安装好了？例如，提供了一个安装测试规程吗？

20.2.12 任务 12：设计/脚本化其他类型的系统测试

作为上述系统测试的补充，可能还需要进行下列系统测试。

- API测试——验证系统正确地使用了API，如操作系统调用。
- 通信测试——验证系统的通信和网络。
- 配置测试——验证系统在不同的系统配置（如软件、硬件和网络等）下能够正确地运行。
- 数据库测试——验证数据库完整性、业务规则、权限和刷新能力等。
- 降级系统测试——验证系统在运行能力受限的情况下（比如说类似于线路连接失败等环境中）能正确地运行。
- 灾难恢复测试——验证系统的恢复规程能够正确运行。
- 嵌入式系统测试——验证系统在低级设备（如视频芯片等）上的运行情况。
- 设备测试——验证需求中定义的所有设备都得到满足。
- 现场测试——验证系统在真实环境下能正常运行。
- 中间件测试——验证中间件能够正确运行，例如客户和服务器的通用接口和可访问性。

- 多媒体测试——使用视频、图片和声音等验证多媒体系统特性。
- 联机帮助测试——验证系统的在线帮助特性能够正常运行。
- 可操作性测试——验证系统在实际的业务环境中能够正常运作。
- 打包测试——验证软件的安装包能够正确运行。
- 并行测试——验证系统的新旧版本中相同的功能运行结果是一样的。
- 移植测试——验证系统能够在不同操作系统和计算机上正确运行。
- 规程测试——验证非自动的规程能够正确运行，例如操作、数据库管理等类似的事情。
- 生产测试——验证系统在实际进行的生产中能够正确运行而不是只能在测试实验室的环境中运行。
- 实时测试——验证一个具有响应时间方面需求的、时间因素非常关键的系统。
- 可靠性测试——验证系统在预定义的期望故障持续时间之内能够正确运行，例如平均故障时间（MTF）。
- 可服务性测试——验证系统的服务设备运作正常，例如，调试一个缺陷和维护规程的平均时间。
- SQL测试——验证查询、数据检索和更新等。
- 存储测试——验证系统的存储需求被满足，例如，溢出文件的大小和使用的主存或辅助存储器的数量。

20.3 步骤 3: 系统测试的评审和批准

20.3.1 任务 1: 评审的日程安排/执行

在实际进行系统测试的评审之前，应该首先安排好测试计划评审的时间，同时保证所有参与人员都有测试计划的最新版本。

像其他访谈和评审一样，系统测试的评审也应该包含如下几个必需的部分。首先要定义讨论的内容，第二步讨论细节问题，第三步是总结，最后是时间安排。评审人员应该预先规定评审的预计持续时间，并且制定一条基本原则，如果规定的时间到了而议程表中的项目没有全部讨论完，就需要再安排一次后续的评审。

本项任务的目的是让开发人员和项目负责人都能同意并接受系统测试计划。如果评审过程中有任何对测试计划的改进建议，都应合并到测试计划中去。

20.3.2 任务 2: 获得批准

获得批准在测试工作中是非常重要的，因为这可以保证测试人员、开发人员和项目发起人三者之间达成必要的、一致的协定。获得批准的最佳方法是通过正式的系统测试计划的签署手续。如果有的话，使用管理审批签署表。如果没有正式的一致承认的手续，则向每位关键的参与人员发送一份备忘录，这些人员至少包括项目经理、开发经理和项目负责人。在文档中要附上最新的测试计划并指出他们反馈的所有意见都被合并进来了，而且如果没有得到反馈，则默认他们同意这一测试计划。最后，要指出在螺旋式开发环境中，系统测试计划是会随着每次迭代过程而演进，

我们会把任何修正都包含进来。

20.4 步骤 4：执行系统测试

20.4.1 任务 1：对系统测试中的修正进行回归测试

本项任务的目的在于重新测试在这一版系统对那些在以前的系统测试周期中发现的缺陷。使用的技术是回归测试。回归测试是一项用来检测由软件修改或修正引起的欺骗性错误的技术。

在软件的整个生命周期中，必须要持续维护一组测试用例，使其一直可用。这组测试用例必须足够完整以便软件的所有功能都被测试，这涉及如何确定测试用例和在上一个测试螺旋中发现的缺陷之间的对应关系的问题，再测试矩阵是一个很好的解决方法。

如上所述，再测试矩阵将测试用例与功能（或程序单元）关联起来。矩阵中的一个检查项表明，当为了增强或改正而修改某个功能（或程序单元）时，对应的测试用例应该被重新测试。空项代表这个测试不需要重新测试。再测试矩阵应该在第一次螺旋开始前建立好，在后面的螺旋中还需要不断地维护。当功能（或程序单元）在一次螺旋中被修改了，应该在再测试矩阵中创建相应的测试用例或者检查已经存在的相应测试用例，为下一次螺旋做好准备。随着后面螺旋过程的进行，一些功能（或程序单元）可能最近都没有进行修改，变得很稳定。在测试螺旋之间应当考虑有选择地从再测试矩阵中删除这些稳定的功能的检查项。

20.4.2 任务 2：执行新的系统测试

本项任务的目的是执行在上一次系统测试周期的末尾创建的新系统测试。在上一次螺旋中，测试团队更新了功能测试/图形用户界面测试、阶段性系统测试和验收测试，为当前的测试螺旋做好了准备。在本项任务中，这些测试将被执行。

20.4.3 任务 3：将系统缺陷记录在文档中

在系统测试执行的过程中，测试的结果必须在缺陷跟踪数据库中报告。这些缺陷一般与一些单独的测试相关联。然而，正式的测试用例的一些变体经常能够发现其他一些缺陷。本项任务的目的在于生成完整的缺陷记录。如果已经正确地记录下了执行的步骤，那么缺陷也就被记录在缺陷跟踪数据库中了。如果缺陷已经记录下来了，则这个步骤的目的就变成收集整理缺陷信息。

根据测试执行的方法的不同，可以使用不同工具来整理和记录缺陷。如果在纸上记录缺陷，那么整理工作将包括收集并整理这些纸介文件。如果缺陷是用电子文件记录的，那么通过搜索可以很容易找到重复的缺陷。

[251]

[252]

验收测试是用户执行的测试，用以证明应用程序符合最初的业务目标并且满足系统的需求。通常验收测试是由系统测试的子集构成（见图21-1）。图中是关于验收测试的讨论，包括如何准备验收测试，如何设计验收测试并将其脚本化，如何执行验收测试，以及如何报告在测试过程中发现的异常等方面的内容。

21.1 步骤 1：完成验收测试计划

21.1.1 任务 1：确定验收测试类型

在本项任务中，最初的验收测试类型清单将被细化，并且选择实际要进行的测试。

验收测试是一种由用户执行的可选测试，用来证明应用程序具备满足用户需求的能力。这个测试的出发点是证明功能而不是发现缺陷，也就是要证明这个系统能够工作。测试的重点少部分在技术问题上，大部分在“对于最终用户来说，系统是否具有较好的业务适用性”这个问题上。由于一般验收测试是由用户完成的，所以有时候用户也会定义一些“特殊的测试”，例如高强度压力测试或容量测试，挑战系统的极限，甚至有时超过我们在系统测试中所进行的测试。

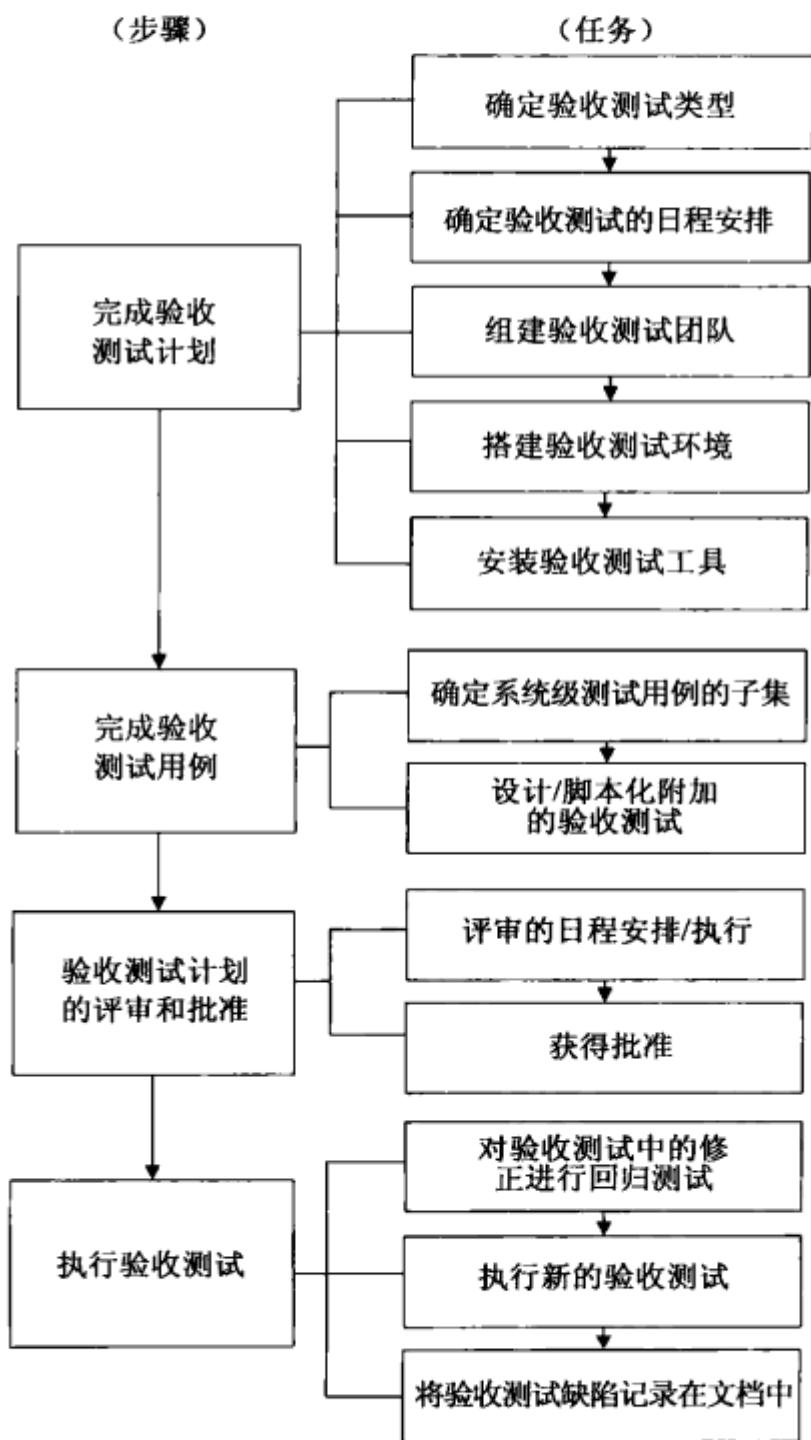


图21-1 进行验收测试（步骤/任务）

21.1.2 任务 2：确定验收测试的日程安排

在本项任务中，要最终决定验收测试的日程安排，包括测试步骤（甚至还可能包括任务）、目标开始日期、目标结束日期和人员职责。本项任务还应该描述验收测试如何被评审、追踪和批准。对验收测试而言，测试团队通常由用户代表组成，但测试环境和测试工具可能和系统测试中使用的没有什么不同。表21-1所示是一个验收测试日程安排的例子。

表21-1 验收测试日程安排

测试步骤	开始日期	结束日期	负责人
总体启动			
组建验收测试团队	8/1/04	8/7/04	Smith, 测试经理
搭建验收测试环境	8/8/04	8/9/04	Smith, 测试经理
安装验收测试工具	8/10/04	8/10/04	Jones, 测试人员
验收测试			
设计/脚本化测试	12/11/04	12/15/04	Jones、Baker（用户），测试人员
测试评审	12/16/04	12/16/04	Smith、测试经理
执行测试	12/17/04	12/22/04	Jones、Baker（用户），测试人员
重新测试验收缺陷	12/23/04	12/25/04	Jones、Baker（用户），测试人员

21.1.3 任务 3：组建验收测试团队

验收测试团队负责设计和执行验收测试、评价测试结果、使用缺陷跟踪系统向开发团队报告发现的所有缺陷。当开发团队修正了缺陷之后，测试团队需要对这些缺陷重新进行测试，以确认修正是正确无误的。验收测试团队中通常应该有用户群体的代表，因为这是他们验收整个系统的最后机会。

255 验收测试团队由一个测试经理领导，其职责包括：

- 组建测试团队；
- 搭建测试环境；
- 组织建立测试的策略、规程和标准；
- 确保测试准备就绪；
- 运行测试计划并对项目进行控制；
- 对各项测试成本进行追踪；
- 确保测试文档的准确性和及时性；
- 管理团队成员。

21.1.4 任务 4：建立验收测试环境

在本项任务中，要最终决定验收测试的环境。一般来说，验收测试的测试环境与系统测试的环境是一样的。搭建测试环境的目的是为测试活动提供一个物理框架。对这个任务而言，测试环境需要在实现前建成并通过评审。

用户验收测试一般是由业务部门执行的,因此,传达给用户的验收测试环境的细节是非常重要的。

21.1.5 任务 5: 安装验收测试工具

在本项任务中要安装验收测试工具并检验以确保其准备就绪。通过运行所需的测试用例和脚本确保测试工具已经可用于实际的验收测试。一般来说,验收测试工具与系统级测试工具是一样的,但这需要业务部门与质量保证部门之间确认。下面是需要考虑的另外一些关于工具就绪方面的因素。

- 培训测试团队如何使用工具。
- 工具与操作环境的兼容性。
- 足够的硬盘空间以满足工具使用的需要。
- 最大限度发挥的工具潜能。
- 厂商的工具帮助热线。
- 修改测试规程使之适用于测试工具。
- 安装工具的最新更新包。
- 确保厂商遵守契约条款。

21.2 步骤 2: 完成验收测试用例

在这个步骤中要设计验收测试用例并将其脚本化。概念上的验收测试用例被转换为可复用的测试脚本,同时创建测试数据。为了协助进行测试用例脚本的开发,可以使用E.7节中的基于图形用户界面的功能测试矩阵模板来记录验收级测试用例,注意把标题中的“功能”两个字替换成系统测试的名字。

256

21.2.1 任务 1: 确定系统级测试用例的子集

一般而言,验收测试用例是由最终用户开发的(当然也不总是这样),通常软件的设计者不应该承担这个职责,因为验收测试是要将系统的实现与其最初的用户需求加以比对。它是为用户接受或拒绝该系统提供的最终测试。最终用户提供测试资源并运行他们自己的测试。他们可能使用系统测试时用的测试环境也可能不使用,这通常取决于测试是否会在最终用户的环境下进行。本书推荐在最终用户的环境下进行验收测试。

典型的验收测试由系统测试的子集组成,这些测试已经在系统测试中设计好了。因此,本项任务包括了标识出将在验收测试中使用的系统级测试。

21.2.2 任务 2: 设计/脚本化附加的验收测试

在验收测试中,除了要复用系统级测试之外,可能还需要对这些测试的某些特定条件进行“扭曲”,以获得系统的最大可接受性。例如,验收测试可能在一个段时间内持续一个特定的吞吐量以测试响应时间的极限。比如,每个小时处理一万件事务,要求平均响应时间为三秒钟,并且要

求90%的事务能在两秒钟或者两秒钟之内完成；再比如，随便从街上找来一个独立的用户，让他坐下使用这个系统和文档，以验证他能有效地使用这个系统。

用户也可能设计一些没有在系统测试中设计过的测试用例。相对于开发人员而言，用户设计的这些测试用例可能更加直观明显，因为用户了解业务需求，并且对于业务操作十分熟悉。这样的测试可能发现一些只有用户能够看到的缺陷。这也帮助用户为实际的安装和生产做好了准备。

验收测试设计甚至可能还包括用户实际数据的使用，因为如果测试在用户看来是更真实的，那么测试结果就更容易得到肯定。而且一些罕见的运行情况若不使用实际数据则可能无法发现。

21.3 步骤3：验收测试计划的评审和批准

21.3.1 任务1：评审的日程安排/执行

在实际进行验收测试计划的评审之前，应该首先安排好评审的时间，同时保证所有参与人员都有测试计划的最新版本。

像其他访谈和评审一样，系统测试的评审应该包含如下几个必需的部分。首先要定义讨论的内容，第二步讨论细节问题，第三步是总结，最后是时间安排。评审人员应该预先规定评审的预计持续时间，并且制定一条基本原则，如果规定的时间到了而议程表中的项目没有全部讨论完，就需要再安排一次后续的评审。

本项任务的目的是让开发人员和项目负责人都能同意并接受验收测试计划。如果评审过程中有任何对测试计划的改进建议，都应合并到测试计划中去。

21.3.2 任务2：获得批准

获得批准在测试工作中是非常重要的，因为这可以保证测试人员、开发人员和项目负责人三者之间达成必要的、一致的协定。获得批准的最佳方法是通过正式的验收测试计划的签署手续。如果有的话，使用管理审批签署表。如果没有正式的一致承认的手续，则向每位关键参与人员发送一份备忘录，这些人员至少包括项目经理、开发经理和项目负责人。在文档中要附上最新的测试计划并指出他们反馈的所有意见都被合并进来了，而且如果没有得到反馈，则默认他们同意这一测试计划。最后，要指出在螺旋式开发环境中，系统测试计划是会随着每次迭代过程而演进，我们会把任何修正都包含进来。

21.4 步骤4：执行验收测试

21.4.1 任务1：对验收测试中的修正进行回归测试

本项任务的目的在于重新测试在这一版系统中重新对那些在以前的验收测试周期中发现的缺陷。使用的技术是回归测试。回归测试是一项用来检测由软件修改或修正引起的欺骗性错误的技术。

在软件的整个生命周期中，必须要持续维护一组测试用例，使其一直可用。这组测试用例必须足够完整以便软件的所有功能都被测试，这涉及如何确定测试用例和在上一个测试螺旋中发现

的缺陷之间的对应关系的问题，再测试矩阵是一个很好的解决方法。

如上所述，再测试矩阵将测试用例与功能（或程序单元）关联起来。矩阵中的一个检查项表明，当为了增强或改正而修改某个功能（或程序单元）时，对应的测试用例应该被重新测试。空项代表这个测试不需要重新测试。再测试矩阵应该在第一次螺旋开始前建立好，在后面的螺旋中还需要不断地维护。当功能（或程序单元）在一次螺旋中被修改了，应该在再测试矩阵中创建相应的测试用例或者检查已经存在的相应测试用例，为下一次螺旋做好准备。随着后面螺旋过程的进行，一些功能（或程序单元）可能最近都没有进行修改，变得很稳定。在测试螺旋之间应当考虑有选择地去从再测试矩阵中删除这些稳定的功能的检查项。

21

258

21.4.2 任务 2：执行新的验收测试

本项任务的目的是执行在上一次验收测试周期的末尾创建的新验收测试。在上一次螺旋中，测试团队更新了功能测试/图形用户界面测试、阶段性系统测试和验收测试，为当前的测试螺旋做好了准备。在本项任务中，这些测试将被执行。

21.4.3 任务 3：将验收测试缺陷记录在文档中

在验收测试执行的过程中，测试的结果必须在缺陷跟踪数据库中。这些缺陷一般与一些单独的测试相关联。然而，正式的测试用例的一些变体经常能够发现其他一些缺陷。本项任务的目的在于生成完整的缺陷记录。如果已经正确地记录下了执行的步骤，那么缺陷也就被记录在缺陷跟踪数据库中了。如果缺陷已经记录下来了，则这个步骤的目的就变成收集整理缺陷信息。

根据测试执行的方法的不同，可以使用不同工具来整理和记录缺陷。如果在纸上记录缺陷，那么整理工作将包括收集并整理这些纸介文件。如果缺陷是用电子文件记录的，那么通过搜索可以很容易找到重复的缺陷。

259

参见F.23节，项目完成情况检查表可以用于确认项目的所有关键活动都已完成。

22.1 步骤 1：执行数据精简

22.1.1 任务 1：确保所有的测试均已执行/解决

在本项任务中，测试团队要对测试计划和日志进行检查，以验证执行了所有的测试（见图22-1）。通常，团队通过确保把所有的测试都记录在活动日志上，然后检查日志来确认是否完成了所有测试。当缺陷处于打开状态并且还没有解决的时候，它们需要将这些缺陷分出优先级，并根据优先级来解决缺陷。

22.1.2 任务 2：通过测试编号整理测试缺陷

在本项任务中，测试团队对被记录的测试缺陷加以检查。如果测试已被适当地执行，那么逻辑上是符合以下假设的：除非提交过缺陷测试文档，否则认为已经得到了正确的或是期望的结果。也就是说，如果这个缺陷没有被修正的话，它应当已经被列入测试缺陷日志了。所以测试团队可以假设，除了那些记录在测试日志中的没有修正或没有满意地修正的问题功能，所有其他的功能都应该是能正确运行的。应该根据测试编号对这些缺陷加以整理，以将它们写入合适的矩阵。

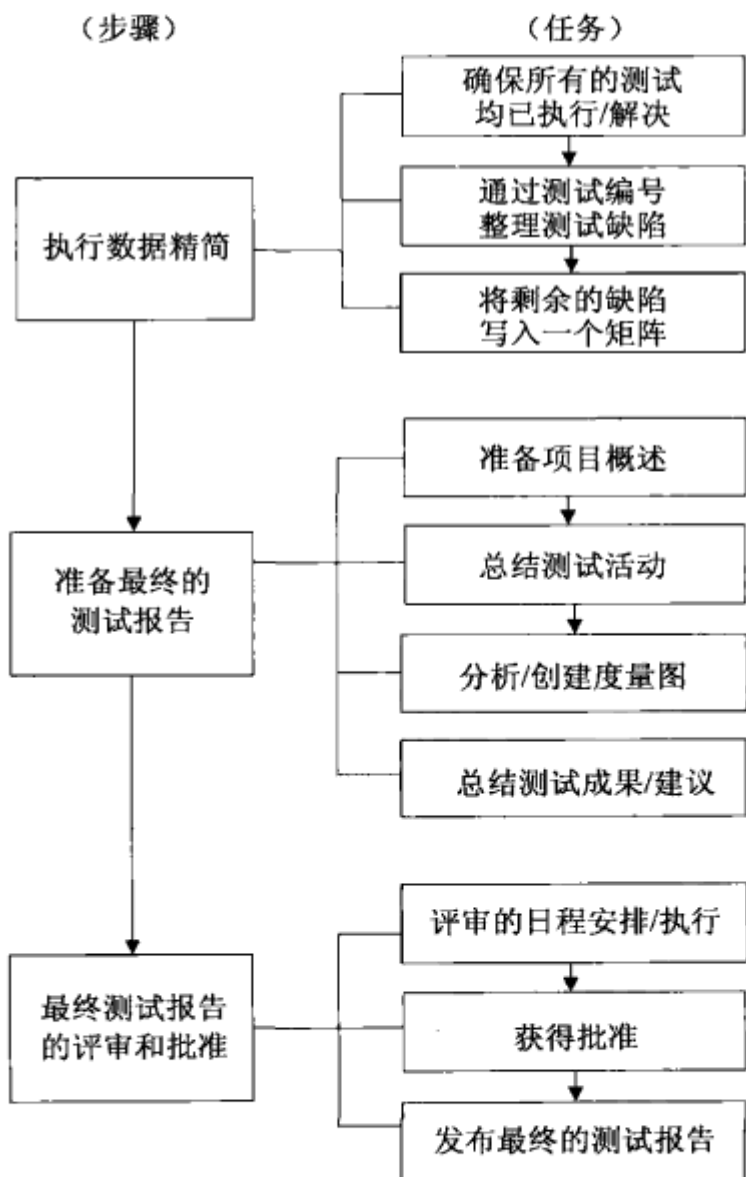


图22-1 总结/报告螺旋测试结果

22.1.3 任务 3: 将剩余的缺陷写入一个矩阵

在本项任务中, 没有修正或没有满意地修正的缺陷应该写入到一个专门的功能测试矩阵中。这个矩阵表明了哪些功能被反复测试了几次。缺陷记录在测试和该测试所测的功能的交叉点上。所有未修正的缺陷都应写入相应的功能/测试矩阵的交叉点上。

22.2 步骤 2: 准备最终的测试报告

最终的螺旋测试报告的目的是从较高的层次上描述测试结果, 不仅仅包括哪些功能能够运行而哪些不能运行, 还包括当应用程序投入生产时, 测试团队根据其性能表现对其做出的评价。

对某些项目而言, 非正式的报告是实践中采用的, 但是在另外一些项目中需要非常正式的报告。以下是对两种极端的折中方案, 提供了基本的信息, 并且不需要进行过多的准备 (参见E.15节中的螺旋测试总结报告, 也可参见E.29节中的最终测试总结报告, 这个报告可以作为具有测试项目关键结论的最终报告)。

22.2.1 任务 1: 准备项目概述

本项任务的目标是以图形的格式记录项目的概述。引言中包含的一些相关信息包括项目名、项目目标、系统类型、目标用户、参与项目的组织、系统开发原因、所涉及的子系统、系统的主要和次要功能以及哪些功能是超出项目范围的, 不会被实现。

22.2.2 任务 2: 总结测试活动

本项任务的目的是为项目进行测试行为的描述, 包括下列信息。

- 测试团队——测试团队的组成, 例如, 测试经理、测试主管和测试人员; 测试团队中每个人的贡献, 例如测试计划、测试设计、测试开发和测试执行等。
- 测试环境——物理测试设施、测试技术、测试工具、软件、硬件、网络、测试库和支持软件等。
- 测试的类型——螺旋 (有几个螺旋周期)、系统测试 (测试的类型以及有多少种) 以及验收测试 (测试的类型以及有多少种)。
- 测试进度 (主要的里程碑)——外部的和内部的。外部的里程碑是指那些项目之外但对项目有着直接影响的事件。内部的里程碑是指在项目内的、某种程度上可以被控制的事件。
- 测试工具——所使用的测试工具及使用它们目的。例如, 路径分析、回归测试、负载测试等。

22.2.3 任务 3: 分析/创建度量图

在本项任务中, 对缺陷和项目中测量的测试管理度量进行收集和分析。为了使工作更有效率缺陷跟踪应该自动化。报告得以处理, 度量结果的总数和趋势得以分析。这样的分析有助于测定

系统的质量以及它投入使用的可接受性，并且对于未来的测试工作也是十分有用的。最终测试报告应当包括一系列的度量图表。下面是建议使用的图表。

1. 按功能分类缺陷

表22-1显示了在每个功能模块或功能模块组中发现的缺陷数目和比例。这一分析将标识出具有最多缺陷的功能模块。一般来说，这类功能的需求和设计都很差。在下面的例子中，报表这一项具有缺陷总数43%的缺陷，表明了这一部分在发布后还需要检查其可维护性。

表22-1 按照功能分类归档的缺陷

功 能	缺陷数目	占总数百分比
订单处理		
创建新订单	11	6
完成订单	5	3
编辑订单	15	8
删除订单	9	5
小计	40	22
客户处理		
创建新客户	6	3
编辑客户	0	0
删除客户	10	6
小计	16	9
财务处理		
接收客户付款	0	0
存储付款	5	3
支付厂商	9	5
填写支票	4	2
显示记录	6	3
小计	24	13
库存处理		
获得厂商产品	3	2
维护库存	7	4
处理退单	9	5
审计库存	0	0
调整产品价格	6	3
小计	25	14
报表		
创建订单报表	23	13
创建应收账款报表	19	11
创建应付账款报表	35	19
小计	77	43
总计	182	100

2. 按测试人员分类缺陷

表22-2显示了在项目中每一个测试人员所发现的缺陷数目和比例。这一分析将标识出测试人员中谁记录的缺陷数目比预计的要少。但是这些统计数据要小心使用。测试人员记录的缺陷少可能是因为他测试的功能区域具有相对较少的缺陷。例如，表22-2中的Baker。另一方面，记录的缺陷数目占总数百分比比较高的测试人员可能是工作效率较高的，例如测试员Brown。

表22-2 按照测试人员分类记录的缺陷

测试者	缺陷数目	占总数百分比
Jones	51	28
Baker	19	11
Brown	112	61
总和	182	100

3. 缺陷差距分析

图22-2显示了在整个项目中已发现的缺陷数和已更正的缺陷数之间的差距。在项目完成的时候，这两条曲线应当能够重合，表明大部分已发现的缺陷都得到了更正，系统已经准备好投产。

4. 缺陷严重程度状况

图22-3显示了在整个项目中3种严重级别（如严重的、主要的和次要的）的缺陷的分布情况。如果严重缺陷占很高的比例，就说明应用程序的设计或架构存在问题，当产品发布投入生产之后需要检查其可维护性。

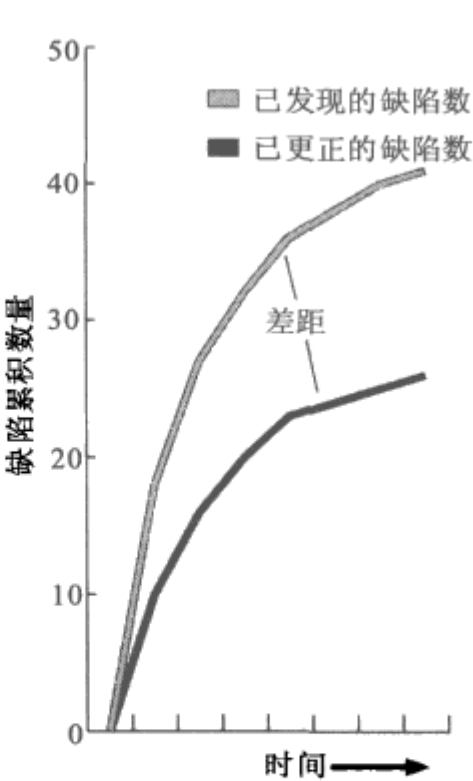


图22-2 缺陷差距分析

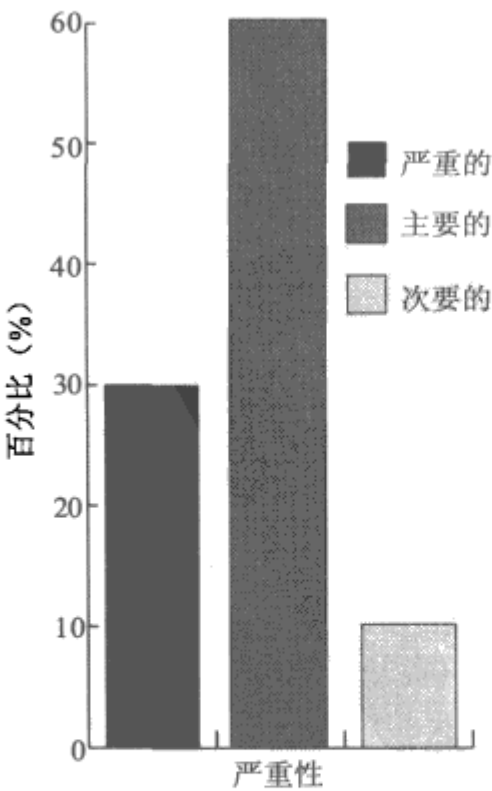


图22-3 缺陷严重性情况

5. 测试零缺陷跟踪

图22-4显示了在整个项目中已发现缺陷的速度，是一个非常有价值的关于测试完成情况的指示器。缺陷的累积数量（也就是测试执行过程中当时的总数）以及某个时间段内的缺陷数量等数据有助于预测从何时开始新发现的缺陷数量越来越少。标志是累积曲线发生“弯折”而且时间段内发现的缺陷数为零。

6. 根本原因分析

图22-5显示了缺陷的来源，例如，架构缺陷、功能缺陷、易用性缺陷等。如果绝大部分是架构缺陷，那么问题将影响整个系统，系统需要进行大幅度的重新设计和重新开发。在产品发布投入生产之后需要检查占总数百分比比较高的缺陷功能的可维护性。

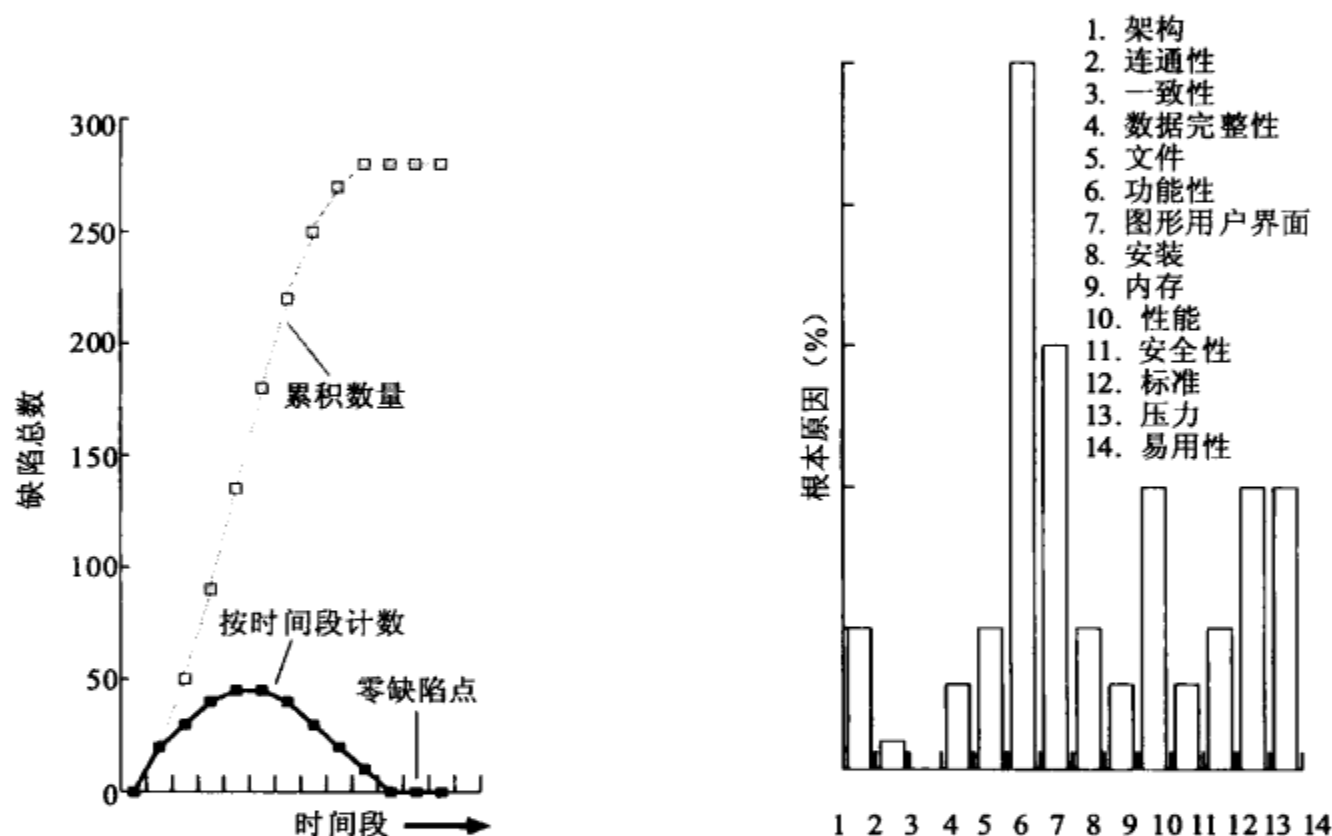


图22-4 测试零缺陷跟踪

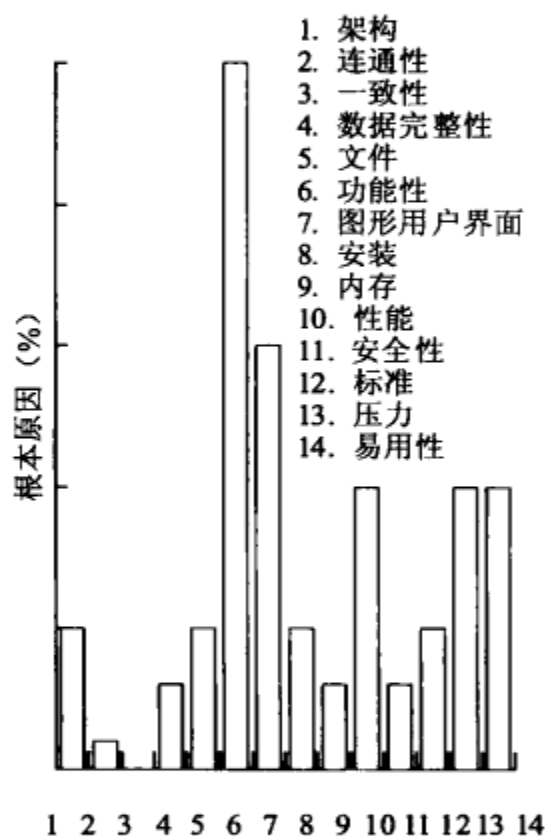


图22-5 根本原因分析

7. 按发现方式分类缺陷

图22-6显示了缺陷都是如何被发现的，例如，通过外部客户、手工测试和其他类似方式。如果通过审查、走查和JAD发现的缺陷占总数百分比不高，说明对这些方面测试关注过多，而对评审过程的重视力度不够。在手工测试和自动化测试之间的百分比差异也证明了自动化测试在测试过程中的贡献。

8. 按发现者分类缺陷

图22-7显示了缺陷都是谁发现的，例如，外部客户、开发部门、质量保证测试等。对大部分项目而言，大部分缺陷都是在质量保证测试过程中发现的。但是，如果是外部或内部客户发现了绝大部分缺陷，则表明质量保证测试不足。

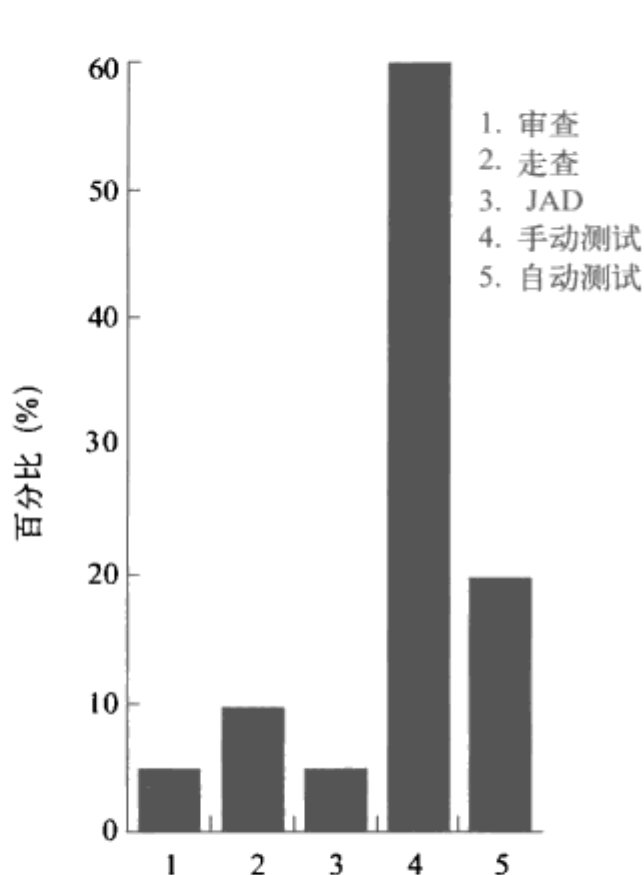


图22-6 按照发现方式分类缺陷

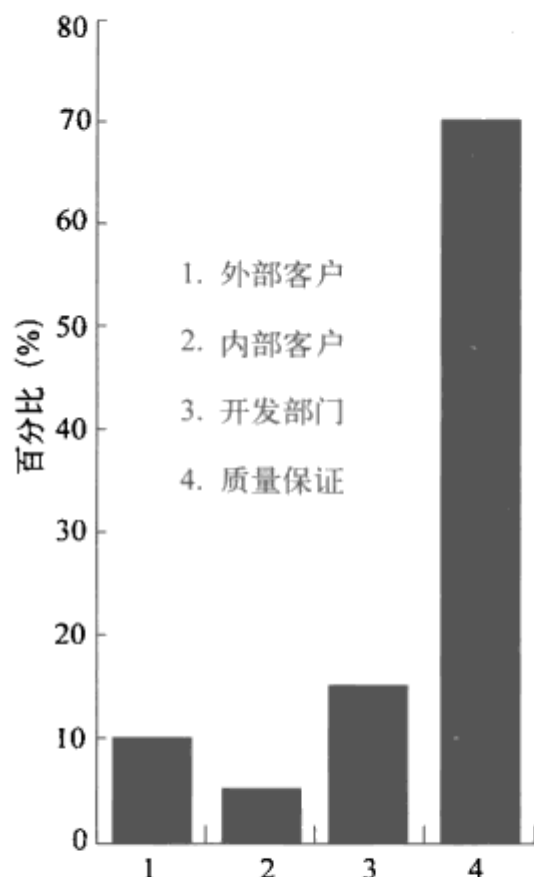


图22-7 按照发现者分类缺陷

9. 功能已测试/未测试

图22-8显示了测试的最终状态,并验证所有或大部分缺陷已经得到修正,系统已经可以投入生产。在项目结束的时候,所有的测试用例都应当已完成,运行时发现错误的测试用例和没有运行的测试用例的百分比应当为零。如有例外,应由管理者对其加以评估,并记录在文档中。

10. 系统测试发现的缺陷类型

系统测试由一个或多个基于系统初始目标的测试组成。图22-9显示了按照系统测试类型分类的缺陷的分布情况。在这个例子中,性能测试发现的缺陷最多,其次是兼容性测试,再次是易用性测试。性能测试在图中占百分比过高的情况表明所设计的系统较差。

11. 验收测试类型发现的缺陷

验收测试是一种由用户来执行的可选测试,用来证明应用程序能满足用户的需求。这个测试的出发点是正面的而不是负面的,比如证明这个系统能够工作。重点不在于对技术问题的关心,

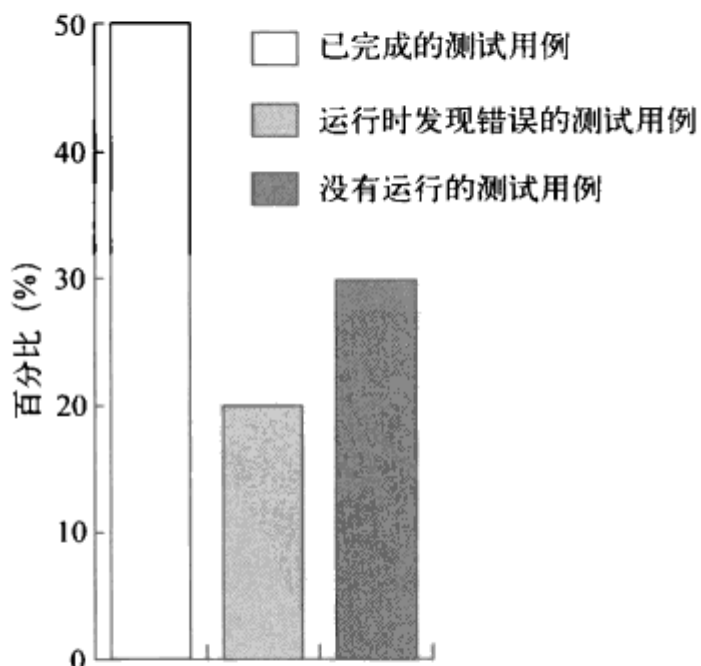


图22-8 功能已测试/未测试

而是从业务角度来说系统对于最终用户来说是否适合。

在验收测试中不应该发现许多缺陷，因为这些缺陷在系统测试的过程中应该都已经得到了修正。在图22-10中，性能测试仍是发现缺陷最多的一类，其次是压力测试，再次是容量测试。

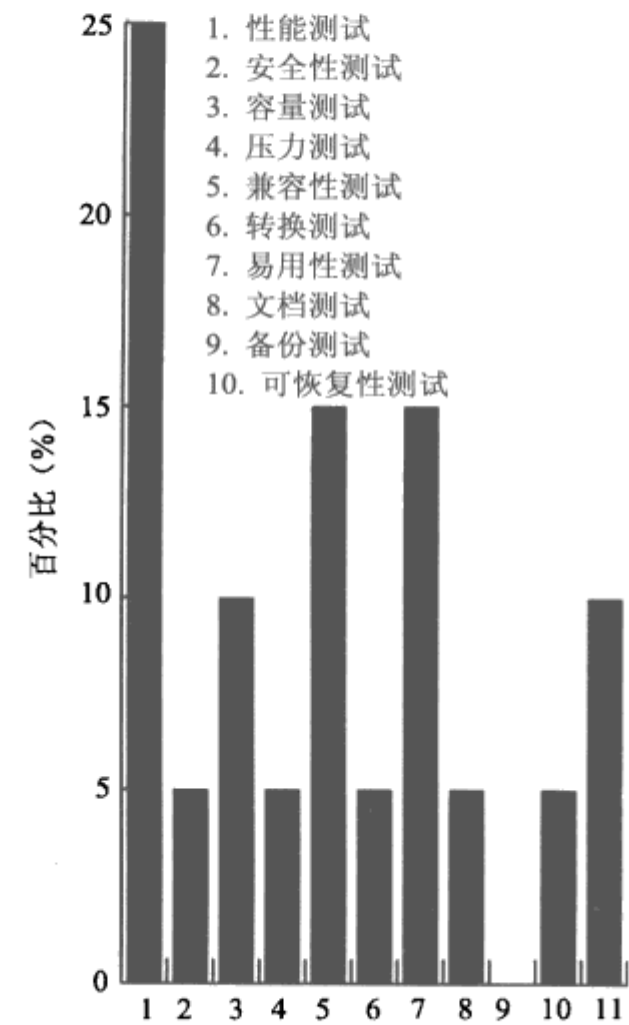


图22-9 系统测试类型发现的缺陷

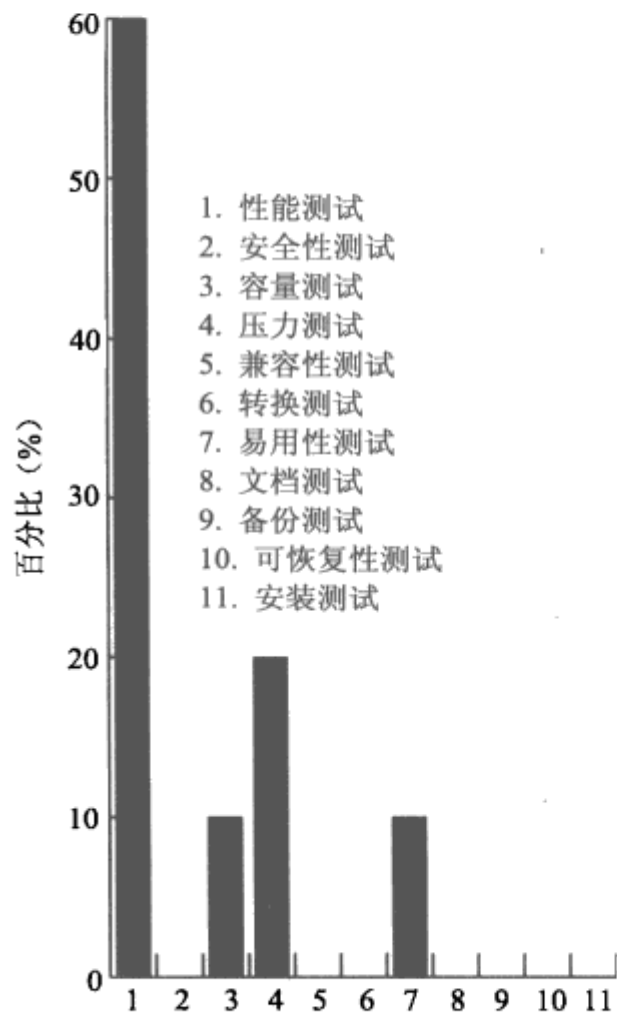


图22-10 验收测试类型发现的缺陷

22.2.4 任务 4：总结测试成果/建议

调查结果给出想要的结果和实际结果之间的差异。建议则是对如何修正一个有缺陷或者提高系统性能提出忠告。测试团队给出的调查结果和建议组成了测试报告的主要部分。

这个任务的目的是根据测试过程和文档中记录的经验教训给出调查结果和建议。首先，数据精简和调查结果其实是一回事，只是这些数据应该以适当的形式给出，以适合项目团队和管理者使用。

测试团队应该为修正一种问题情况做出建议。项目团队则应该确认调查结果的正确性和建议的合理性。每个调查结果和建议都可以被记录在表22-3显示的调查结果/建议矩阵中。

表22-3 调查结果/建议矩阵

调查结果描述 ^a	业务功能 ^b	影响 ^c	对其他系统的影响 ^d	修正费用 ^e	建议 ^f
最初给项目分配的测试人员不够	N/A	造成测试过程比初始日程安排滞后	N/A	从代理机构再雇用5名测试人员	在未来的项目计划中增加更多的资源
开发部门对缺陷跟踪的监视不足	N/A	显著缺陷的数目明显增长	N/A	让开发部门加班	在未来的项目中质量保证部门要经常强调缺陷跟踪的重要性
自动化测试工具对回归测试确实做出了显著贡献	N/A	提高了测试生产力	N/A	N/A	要尽量多地利用测试工具
在一个功能区域中出现了数量过多的缺陷	报表	导致了开发人员返工的时间	N/A	开发人员额外的加班	在项目早期要多进行对技术设计的评审
功能区域与其他系统不兼容	订单处理	返工费用	不得不重新设计数据库	雇用了一个Oracle数据库管理员	在项目早期要多进行对数据库设计的评审
30%的缺陷是极严重的	N/A	显著地影响了开发和测试工作	N/A	雇用了额外的开发程序员	在项目早期要多进行技术评审,并且签署规程要更加严格
功能/图形用户界面的缺陷最多	N/A	需要进行不少返工	N/A	让测试人员加班	在项目早期要多进行技术评审,并且在签署规程要更加严格
由于性能负载测试工具不能正常运行,两个测试用例无法完成	用1000个终端对订单输入进行压力测试	在极端的压力条件下无法保证系统能够正常运行	N/A	推迟交付系统的时间,直到获得需要的测试工具(推迟2个月,收入损失85 000美元,购买工具花费10 000美元)	收入的损失使交付系统的风险显得不那么重要了。交付系统,获取性能测试工具并完成压力测试

注: a. 这包括了对缺陷跟踪数据中记录的缺陷信息的问题描述,也可以包括测试团队、测试规程或者测试环境的调查结果和建议。

b. 描述了涉及和受影响的业务功能。

c. 描述了调查结果对操作系统可能造成的影响。影响应该或者被描述为主要的(缺陷将导致应用系统产生不正确的结果)或者被描述为次要的(系统是有问题的,但是结果是正确的)。

d. 描述了除当前被测系统之外调查结果还将在何处影响其他的应用程序系统。如果调查结果影响到了其他的开发团队,他们也应当参与决定是否对这个问题加以修正。

e. 在决定安装一个还有问题没有修正的系统之前,管理层必须了解其代价和收益。

f. 描述了测试团队给出的采取何种行动的建议。

22.3 步骤 3: 最终测试报告的评审和批准

22.3.1 任务 1: 评审的日程安排/执行

在实际进行测试总结报告的评审之前,应该首先安排好评审的时间,同时保证所有参与人员

都有测试计划的最新版本。

像其他访谈和评审一样，测试总结报告的评审应该包含如下几个必需的部分。首先要定义讨论的内容，第二步讨论细节问题，第三步是总结，最后是时间安排。评审人员应该预先规定评审的预计持续时间，并且制定一条基本原则，如果规定的时间到了而议程表中的项目没有全部讨论完，就需要再安排一次后续的评审。

本项任务的目的是让开发人员和项目负责人都能同意并接受测试报告。如果评审过程中有任何对测试报告的改进建议，都应合并到报告中去。

22.3.2 任务 2：获得批准

获得批准在测试工作中是非常重要的，因为这可以保证测试人员、开发人员和项目负责人三者之间达成必要的、一致的协定。获得批准的最佳方法是通过正式的测试计划的签署手续。如果有的话，使用管理审批签署表。然而，如果没有正式的一致承认的手续，则向每位关键的参与人员发送一份备忘录，这些人员至少包括项目经理、开发经理和项目负责人。在文档中要附上最新的测试计划并指出他们反馈的所有意见都被合并进来了，而且如果没有得到反馈，则默认他们同意这一测试计划。最后，要指出在螺旋式开发环境中，测试计划是会随着每次迭代过程而演进，我们会把任何修正都包含进来。

22.3.3 任务 3：发布最终测试报告

测试报告将在合并了评审的建议之后给出最终版本，并分发给相应各方。本项任务具有短期目标和长期目标。

短期目标是向软件用户提供信息，以决定系统是否已经适合投产。同时，还提供关于一些重要的问题（包括哪些测试没有完成或有显著的缺陷）和建议的信息。

长期目标是从质量的角度来看，为项目提供一些关于项目如何管理和如何开发的信息。如果系统投产后功能出现了问题，项目组可以使用这份报告对问题加以追踪，比如追踪产品使用过程中发现系统功能存在的严重缺陷以及功能不正确的问题。项目组和组织也获得了一个向当前项目学习的机会。通过报告也可以对开发部门、项目管理和测试规程哪些运作良好，哪些运作不好或者需要改进做出一个判定，这对于未来的项目是十分有价值的。

项目管理方法

根据美国质量协会（American Society for Quality, ASQ）的定义，项目管理是指运用各种知识、技能、工具和技术来满足项目需求的过程。后面几章通过描述达到项目质量的基本测试管理过程和组织方法，把项目质量管理实践和方法应用到软件测试中。

本部分的目标如下：

- 定义项目框架；
- 开发产品质量和项目质量之间的依赖关系；
- 刻画项目框架的各个阶段；
- 描述项目范围和产品质量之间的重要关系；
- 定义项目经理和测试经理在质量管理中的作用；
- 描述支持项目质量管理的质量计划过程的步骤；
- 强调影响项目估算的各种因素；
- 探讨支持质量控制的缺陷管理活动；
- 演示缺陷跟踪技术是如何影响项目质量的；
- 介绍把测试与开发方法集成到一组统一过程的好处；
- 展示构建集成方法的步骤；
- 阐述组织结构为什么会影响项目经理和测试经理完成其任务的方式；
- 建立应对组织挑战的方法；
- 描述另外一种测量项目进度和对需求的符合程度的质量量度。

23.1 项目框架

项目框架是将质量过程与项目的各个阶段联合起来，并且将项目质量管理与系统（或软件）开发方法同步起来的一种简单、有效的方法。

项目框架使用了美国项目管理协会的过程组来完成下面两项工作：

- 将质量过程嵌入到项目的各个阶段中；
- 将质量计划、质量保证和控制活动与系统（或软件）开发生命周期（SDLC）的输出相结合。

只有项目经理和测试经理在项目框架的各个阶段协作，项目最终获得的才会是高质量的产品。

关于项目管理协会（PMI）的更多信息请访问www.PMI.org。

23.2 产品质量与项目质量

279

项目经理对最终的产品质量和项目质量负责。产品质量和项目质量的区别可以根据PMI的项目管理知识体系（PMBOK）刻画出来，简述如下。

- 产品质量满足顺应需求和适宜使用的明确准则。
- 项目质量在达成一致的项目范围内，提交所需的产品或服务，并且在没有超出预算的情况下满足经审批的进度。

总而言之，产品质量和项目质量包含了项目质量管理。

23.3 项目框架的组成

图23-1所示的项目框架将PMI的5个项目管理过程组看成包罗万象的项目阶段。这张图意味着灵活性，并假设各个SDLC阶段可能会发生重叠的活动。



图23-1 项目框架

23.4 项目框架与持续质量改进

项目经理和测试经理共同分担持续质量改进的职责。他们用项目框架将持续质量改进融入到SDLC的每个阶段中。

项目框架符合图23-2所示的修订的Deming的“计划-执行-检查-改进”循环。

使用PDCA循环的例子举不胜举，下面就给出几个。

计划。清晰地定义项目范围和产品范围，了解要完成项目目标所需的条件、策略和方法。

执行。为按照批准的项目范围完成工作，创建条件和规程。

必要的时候要给予培训，以便需要的时候具备所需的技能。确保项目团队的成员既了解项目目标，又了解自己的项目工作。

检查。判断是否按照计划完成了项目交付物，获得的结果是否是所期望的。

改进。应对偏差的方法是，调整质量过程来避免与项目质量和产品质量的偏差。具体包括以下几个动作。

- 确认对需求和范围的变更。
- 判断项目交付物是否满足质量保证测量。
- 确保项目文档（如项目进度表和项目预算）被更新。
- 在相应条件下评估变更的影响，来检测和纠正与项目质量标准的偏差。
- 针对任何主要的偏差完成根本原因分析，并调整过程以免重现。

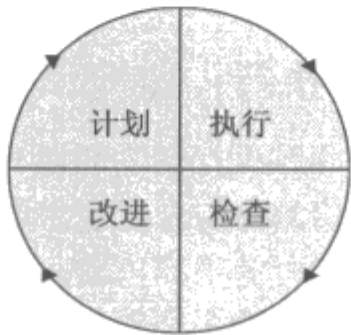


图23-2 “计划-执行-检查-改进”循环

23.5 项目框架的各个阶段

23.5.1 启动阶段

项目启动的标志是项目发起人承诺资助项目。启动活动包括生成项目章程（项目章程总结了产品和项目的大致范围），以及授权项目经理来承担项目领导的职责。早期的需求定义、项目团队动员和利益相关人分析都是启动活动，这些活动为计划阶段奠定了基础。

项目的质量定义在启动阶段开始，通过逐步细化来获得客户对产品验收的详细期望。在启动

阶段编写初步的质量陈述，即初始的质量政策，有助于通过建立客户的验收环境，助推启动测试计划。

23.5.2 计划阶段

为项目做质量管理方法的计划是与开发产品和项目的范围和需求并行进行的。项目假设、依赖关系和风险是测试策略的输入，其他输入还包括应用和架构模型以及集成需求。

计划阶段的开发活动（如指导概念的证明或演示原型）提供了一种机会，来评估项目是否需要现有的质量政策、测试环境、测试工具和测试方法进行修改和增加。

计划阶段获得的基本质量成果是项目的最终质量政策和从功能及非功能需求导出的测试策略。

质量保证和质量控制的计划直到资源分派给工作分解结构（项目进度表）中的质量任务时才算完成。

23.5.3 执行、监视和控制阶段

因为项目执行活动与监视和控制活动是相互依赖的，这个阶段将这些过程结合起来。审查项目工作以检测出与项目需求的偏差，即确保符合项目需求。因为范围定义很好的测试计划可以检测出未经许可的工作内容，在这一阶段还要确认项目范围。

这一阶段的多层测试活动被分布在SDLC的各个阶段中。这项工作由应用开发组织、基础设施组织和测试组织（如果有）来完成。下面时这项工作安排的一个示例。

详细设计

- 完成功能需求和非功能需求的确认。
- 根据应用和系统接口的最终设计确认测试方法。
- 根据系统性能规约确认测试方法。
- 确认客户、应用开发人员和基础设施团队接受质量标准。
- 确认开发和测试环境在构建和测试活动之前满足规约。

构建和测试

- 完成接口（和测试数据转换）的代码测试和单元测试。
- 确定应用报告、集成和性能的测试方法。
- 确定硬件集成和性能的测试方法。
- 为功能需求和非功能需求创建测试计划。
- 根据功能需求和非功能需求创建和确认测试用例。
- 创建用户验收测试计划和测试用例。

客户验收测试和提交准备

- 需要的时候完成应用报告、集成和性能测试来满足质量标准。
- 需要的时候完成硬件集成和性能测试来满足设计标准。
- 需要的时候完成用户验收测试来满足宜用性的定义。
- 完成和签署测试报告。

23.5.4 收尾阶段

项目框架的最后一个阶段的特征是用用户验收签字并且移交产品（成功提交）。

在签字确认项目完成之前，项目经理要完成许多的管理任务。项目收尾的前提条件是，确保针对每个规定的项目质量阈值，所有缺陷已经被解决，并且确保缺陷日志已经完成。尽管如此，项目还是要到得到了所有的测试输出结果才结束，以便这些制品被其他项目引用。

23.6 划定项目范围以确保产品质量

PMI规定，项目范围确认关心工作结果的验收，而质量控制关心工作结果的正确性。因为范围确认在项目定义的最早阶段开始，项目经理要在项目的启动阶段，利用质量控制和范围之间的依赖关系，确认在后续项目阶段中的项目开销。

23.7 产品范围和项目范围

定义产品范围是定义项目范围的前提。它们之间的关联很简单：产品范围描述的是产品（或被提交的服务）的特征，项目范围规定的是交付产品必须做的工作。

产品的特性和功能表现为需求，项目团队评估为了满足每项需求要完成的工作（任务）。

采用主动的项目管理方法，项目经理将工作评估拓展至包括为了确认需求投入的时间和可能的资源。

在启动阶段评估确认活动的好处是，客户、项目经理、测试经理协商项目必须交付的项目质量的验收级别。协商期间，项目经理和测试经理学习产品特性和功能的一般验收标准，阐明项目质量的边界。正式的认可的项目范围通常表现为项目规章。项目范围的详细描述在范围陈述中开发。

23.8 项目规章

项目规章是生动的业务文档，正式承认资助项目。项目规章向项目发起人和客户展示产品范围和项目范围的简单总结，并且授权项目经理动员项目团队。项目规章在项目范围发生变化时也要更新。

项目规章对项目资源进行分类，为资源分配高级的角色和职责。项目规章还为项目利益相关人提供未来交付物（包括测试计划、测试规约和测试结果）的清单。

项目规章应该简单，并且是非技术性的。项目规章通常应该包括下面几部分。

- 范围：
 - 产品范围；
 - 项目范围。
- 利益相关人。
- 项目资源。
- 业务影响。
- 业务目标。

- 项目调整。
- 项目收益。
- 高级交付物。
- 项目方法。

284

23.9 范围陈述

范围陈述是生动的项目文档，这一文档在项目团队开发详细需求时由项目经理更新。在启动阶段，范围陈述包含项目资源和成本的早期评估。

即使是在项目的早期阶段，项目陈述对于项目质量管理还是至关重要的，因为它将项目范围限制在要交付产品必须做的工作。交付产品的工作包含质量保证和质量控制。

各个组织和部门的范围陈述格式会有所不同，但范围陈述通常包含下面几部分。

- 行政总结。
- 背景。
- 业务目标。
- 项目成本。
- 范围：
 - 产品范围；
 - 项目范围；
 - 范围之外。
- 成功标准。
- 依赖关系。
- 假设。
- 约束条件。
- 已知风险。
- 估算的时间安排，包括最初的工作分解结构。
- 范围管理方法。

23.10 项目经理在质量管理中的作用

项目经理负责，在项目利益相关人之间协调和沟通审定的范围变化的影响。项目经理使用范围陈述检查与项目范围的偏差。如果是审定过的，范围偏差不是负的，但是必须分析任何未经审定的变化的影响，并且查出根本原因。

项目范围定义和质量管理的纠结在一起的，以便产品范围中的任何变化都影响项目范围。因为项目范围包括测试产品，所以改变产品范围直接影响到给质量保证和质量控制分配的项目资源。

285

项目经理越早定义产品范围和管理产品范围的方法，他们在资源、次序、成本和项目持续时间的管理方面就越高效。

总之，通过定义和管理项目范围，项目经理作为一种媒介，帮助测试经理验证需求已经定义且是可测试的，验证范围变化已经及时沟通，并且验证产品适合使用。

23.11 测试经理在质量管理中的作用

测试经理负责确保产品达到验收级别，符合功能和非功能需求。项目质量管理层需要确保与需求的符合程度，没有把质量保证和质量控制活动组织成一系列的阶段（混和或补充软件或系统开发生命周期）的情况下，他们几乎什么都不做。

下面的任务描述不必须是顺序的，可以重叠。在某些情况下，这些工作在并行活动中完成。

23.11.1 分析需求

项目框架启动阶段的早期，业务用户开始明确地阐述他们的需求，通过与当前业务过程对比，描述他们期望自己的业务过程将来如何工作。

很多组织根据业务分析来把业务用户的描述转换成业务需求，总结用户对新特性和新功能的期望。因为有经验的业务用户一般关心的是业务过程而不是系统行为，他们倾向于作出系统条件的假设。出于这个原因，业务分析师可能并不检查某些业务用户的期望是否基于对系统条件的假设。如果隐含条件没有被测试，那么测试是不完整的，也不会满足终端用户的需求。

测试经理应该与用户一起评审业务需求，以确定系统的隐含条件。评审最好是在计划阶段编写评审测试策略和测试场景的时候完成。

23.11.2 进行差距分析

测试经理应该在计划阶段的早期开始初步的差距分析，确定需求和规约文档之间的不一致。

只要可能，完善的差距分析跨越计划和执行两个阶段。分析包括基线文档（如用例和系统设计文档）。为业务用户的最终需求提供信心保证时，确定和解决差距是很基本的。

在产品发布到生产环境（加速提交），并且业务场景没有产生期望的结果之后，需求文档和功能或设计规约之间的差距就变得很明显了。差距分析减少了修复追溯到需求和技术文档之间冲突的问题所需的返工。

23.11.3 避免重复

在计划阶段，测试经理首先要确保测试用例是完善的，同时要确保测试用例不重复。如果在计划阶段没有解决这个问题，那么就存在一种风险：针对不同的条件执行相同测试用例的同一个类，可能会增加测试周期持续的时间，放慢测试人员测试未测试的功能的速度。

等价类划分（附录G中描述）是避免重复测试用例覆盖非常有价值的技术。这种技术按照导致同一种处理和输出的输入条件对业务功能进行分类。等价类划分的结果是简洁的测试用例集，增加测试人员定位缺陷的能力。

请注意，由较差的测试计划和测试用例设计引发的冗余测试与重复测试用例不同，冗余测试目的是验证集中在代码具体区域的异常的解决。

23.11.4 定义测试数据

定义测试数据是测试计划阶段的测试计划活动中至关重要的组成部分。测试经理负责确保执行测试用例所需的数据在测试环境中是可用的，并且要确保所有测试用例都用正确的数据集执行。

数据创建的原则要在业务分析师和开发人员的帮助下，在测试计划阶段确定下来。在测试计划中应该确定测试周期的数据集的位置，还要确定需要刷新和恢复数据集的方法和时间。

23.11.5 确认测试环境

测试经理应该在测试策略文档中定义测试环境。这一定义必须是完整的，并且确定了执行测试用例所需的所有接口。此外，在影响测试执行的接口置于测试工程师的控制之外时，测试经理写一个说明来总结这会给测试工作带来的风险。

测试环境定义之后，测试经理要准备一份检查表，验证测试环境像期望的那样工作。这份测试检查表也可以用于在每个测试周期结束时恢复测试环境。正常情况下，最初的测试环境应该在SDLC的详细设计阶段验证。

23.11.6 分析测试结果

测试执行期间，测试经理要负责测试结果，以确定需要更正和澄清的测试场景。

例如，规约文档定义了开始时间、使用期限和结束时间等。测试结果的分析证明了范围测试的结果是正确的，时间范围的测试确认了开始时间不会早于结束时间？

23.11.7 提交质量

测试经理的主要职责是，向业务用户提交与需求的偏差很小的产品，这一产品满足业务用户需要。如果客户接受了这一产品，测试工作就足够了。如果客户接受了这一产品并且测试及时且在预算内，测试工作就是成功的。

23.12 给测试经理的建议

23.12.1 请求别人的帮助

在测试用例开发期间，测试经理和测试团队应该主动请求业务用户和开发人员帮忙确认团队的预期测试结果。协作的好处很多：开发人员了解了测试人员的验证方法，测试人员了解了终端用户的功能定义，测试经理了解了开发过程的业务视角。

23.12.2 出现问题时及时沟通

测试管理要求测试经理、测试开发人员和项目利益相关人之间有效沟通。测试执行阶段出现的问题必须尽可能传达给利益相关人。让利益相关人了解其关系的状态和进度有助于决策者将精力集中在影响质量的问题上。

有效地沟通需要测试经理了解利益相关人的特殊需要，了解他们要做出很好的决策所需的信息类型。例如，测试用例完成的百分比对于不同的利益相关人意味着不同的事情。测试经理有责任知道，哪一类利益相关人会使用统计数据作出决策，哪一类利益相关人会因为统计数据对他们没有价值而忽略统计数据。

23.12.3 不断更新自身的业务知识

软件开发支撑着商业企业。要提升深厚的测试能力，测试经理及其团队必须拓展他们的业务知识。如果没有深厚的业务背景知识，他们就很难说服开发人员或者客户接受在测试中发现的系统缺陷的重要性。

23.12.4 学习新的测试技术和工具

软件产业是一个变化很快的产业。测试经理如果不学习如何应用下一代软件测试技术、测试工具和测试方法，他们的技能很快就会落伍。技能落伍的结果就是不能从使用高效的工具和方法中节约成本。如果测试经理对这种变化很抵触，他就不会站在支持业务按预期发展的立场提高质量，降低预算，缩短进度。

23.12.5 改进过程

测试经理应该承担持续质量改进的职责，一种方法是将已经学到的经验和教训用于整个项目。在项目团队评审用于交付产品的端到端过程的结果时，出现了提升测试效率和产品质量的新概念。

评审生产支持问题将为测试经理提供大量有用的信息，产品发布到生产环境后发现问题，表明测试方法和计划可能有差距。

23.12.6 创建知识库

在不同项目中收集的宝贵经验应该形成文档，以便这些知识可以在其他项目中复用。测试经理应该记录那些在每次测试项目执行中遇到的积极或消极因素，并把这些信息组织起来，以便项目中的其他成员可以复用这些信息。

289

23.13 质量项目管理和项目框架的好处

下面给出将质量项目管理过程和项目框架过程集成在一起的好处。

启动阶段

- 项目范围确定开始，并作为测试计划的输入。
- 建立客户验收的环境。
- 项目经理和测试经理商讨项目质量的验收级别。

计划阶段

- 产品的产品需求与质量管理方法同时开发出来。
- 项目假设、依赖关系和风险是测试策略的输入。

- 计划质量保证测试和用户验收测试，避免不必要的重复工作。
- 使用测试计划确认项目范围，检测出任何不需要的工作。

执行、监视和控制阶段

- 检测出任何未经审定的范围，并且查出根本原因。
- 构建测试活动之前，开发和测试环境满足规约。
- 定稿软件和硬件测试方法。
- 多层测试活动分布到SDLC的各个阶段。
- 强调继续改进，增强产品和项目性能。

收尾阶段

- 解决质量缺陷。
- 客户认可产品适宜使用。
- 获得所有测试输出结果，以便将来参考和复用。

为了了解将质量项目管理过程和项目框架过程集成在一起的好处，测试经理和项目经理应该整合各自的角色和职责以及他们所用的方法。如果他们做到这一点，就会大大增加满足客户对产品质量的期望和项目质量业务目标的可能性。

24.1 项目质量管理过程

项目质量管理囊括了提交符号客户要求的质量级别的项目产品所需的所有工作。PMI把项目质量管理分为以下几个常见的过程。

- 质量计划：计划质量方法。
- 质量保证：定义与需求符合程度的级别，并将持续质量改进结合到测试过程中。
- 质量控制：执行测试，与质量保证过程中定义的质量阈值进行对比，度量测试结果。

这些项目质量管理过程被整合到前面描述的项目框架的各个阶段。下面描述质量计划阶段的计划活动。

291

24.2 质量计划

有经验的测试经理会系统地计划测试策略，选择测试执行方法，指定需要的测试件。与项目经理合作，测试项目经理会提出下列计划目标。

- 定义完成所需测试类型的策略。
- 实现需求和测试用例之间的可追溯性，保证好的测试覆盖。
- 准备测试用例和测试脚本。
- 评审所有的测试文档。
- 计划数据需求和可用性。
- 安排执行进度。

在没有测试经理加入的情况下完成测试活动、成本和进度表计划，整个项目进度表中很少有现实的测试工作时间线。

测试经理经常是项目开始后才加入到项目中。一旦项目已经步入正轨，项目经理就不能做回顾性的测试工作量估算，也不能调整进度。在这种情况下，测试经理必须采用预先定义好的测试进度表。

24.3 确定高级项目活动

当项目范围很清晰并且有文档记录的时候，项目团队确定要交付项目需要完成的所有主要高

级活动。项目团队成员将每个高级活动分解成工作，当满足了下列需求时工作的分解就是充分的。

- (1) 不需要进一步的信息输入就能在短期内完成。
- (2) 产生一个可交付物（可交付物必须有结论）。
- (3) 可以在现实度量的基础上估算。
- (4) 不能进一步被分解为一个人可以完成的活动。

24.4 测试工作量估算

估算测试工作量要考虑完成计划的测试所需的资源的类型和成本。要估算软件测试的成本，测试经理和项目经理必须考虑以下几项内容。

- 测试人员的数量及测试速度。
- 测试人员的经验级别和生产率。
- 支持工作量所需的硬件和软件的成本。
- 公司分配给项目预算的管理费用。

测试经理和项目经理评估影响测试工作量大小的下列因素。

- 为测试执行阶段计划的测试周期的数量。
- 需要测试的接口的数量。
- 测试批处理运行的数量。
- 与策略中相一致的缺陷修复周转时间。
- 所需测试数据的可用性。
- 缺陷管理和解决过程。
- 变更管理过程。

24.5 测试计划

虽然具体百分比会根据项目情况有所不同，但平均而言，测试团队将总工作量的15%花在定义测试条件和准备可追溯矩阵、测试用例、测试脚本、测试数据和执行计划这样的关键任务上。

通常首先用业务需求文档准备和映射测试条件，以确保测试覆盖是完整的。通过建立广泛测试这些条件所需的数据值，测试条件变为测试用例。

推荐的做法是，将整个应用分解为几个模块和子应用，确定构成测试计划核心的测试条件、测试用例和测试脚本。

测试条件和测试用例被细化，并被分类为复杂、中等和简单条件/用例。测试条件/用例的数量和准备测试脚本的时间是测试计划活动的主要组成部分。对条件/用例复杂性的合适级别作出决定需要整个项目团队的技术专长和实用专长。

执行测试范围的工作量——准备测试计划、发布测试策略和评审交付物的时间——累加到测试计划工作量中。

项目经理和测试经理还应该代管日常缺陷会议，会议召集，并参加项目的计划和执行阶段的其他会议。

图24-1和图24-2所示的项目计划范例定义了在一个测试项目中完成的典型任务。

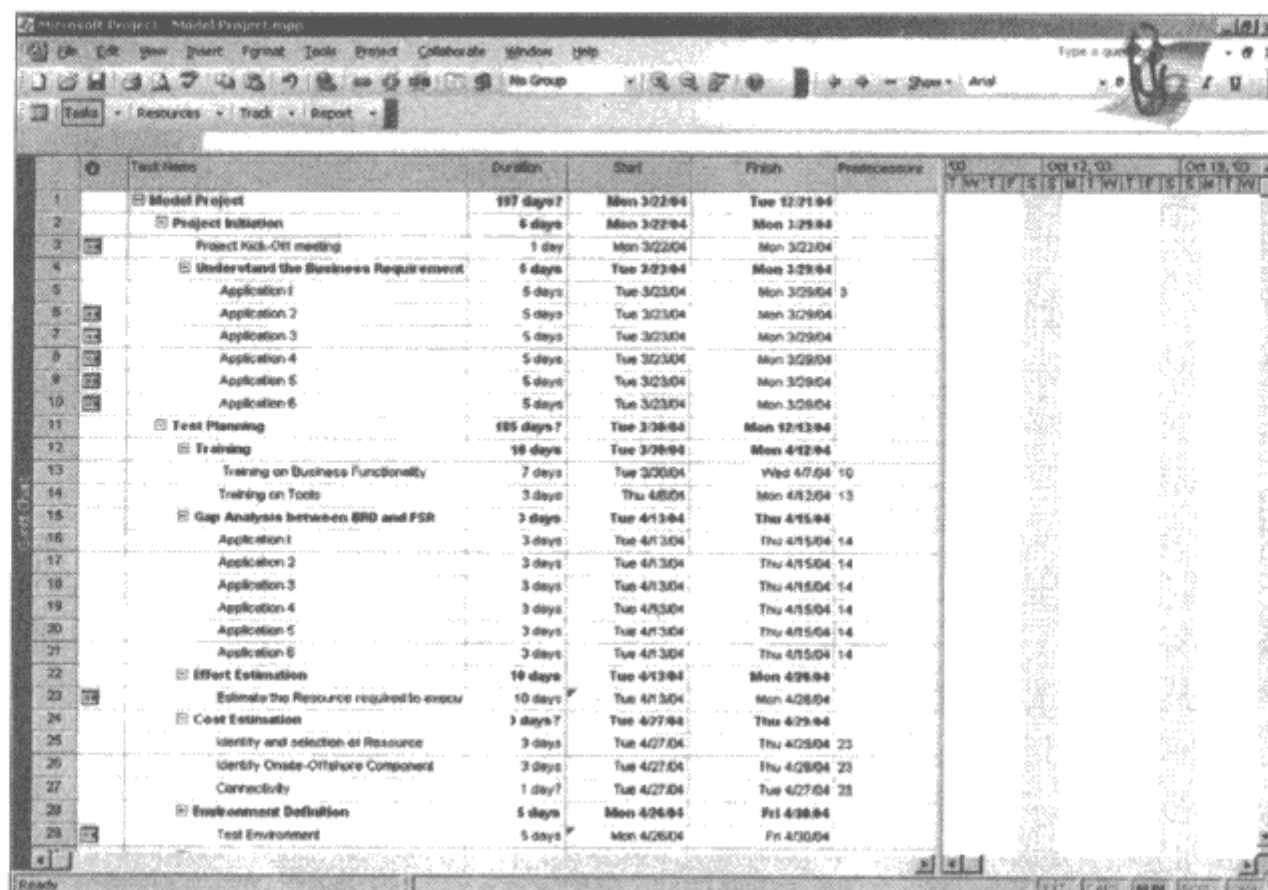


图24-1 测试计划范例

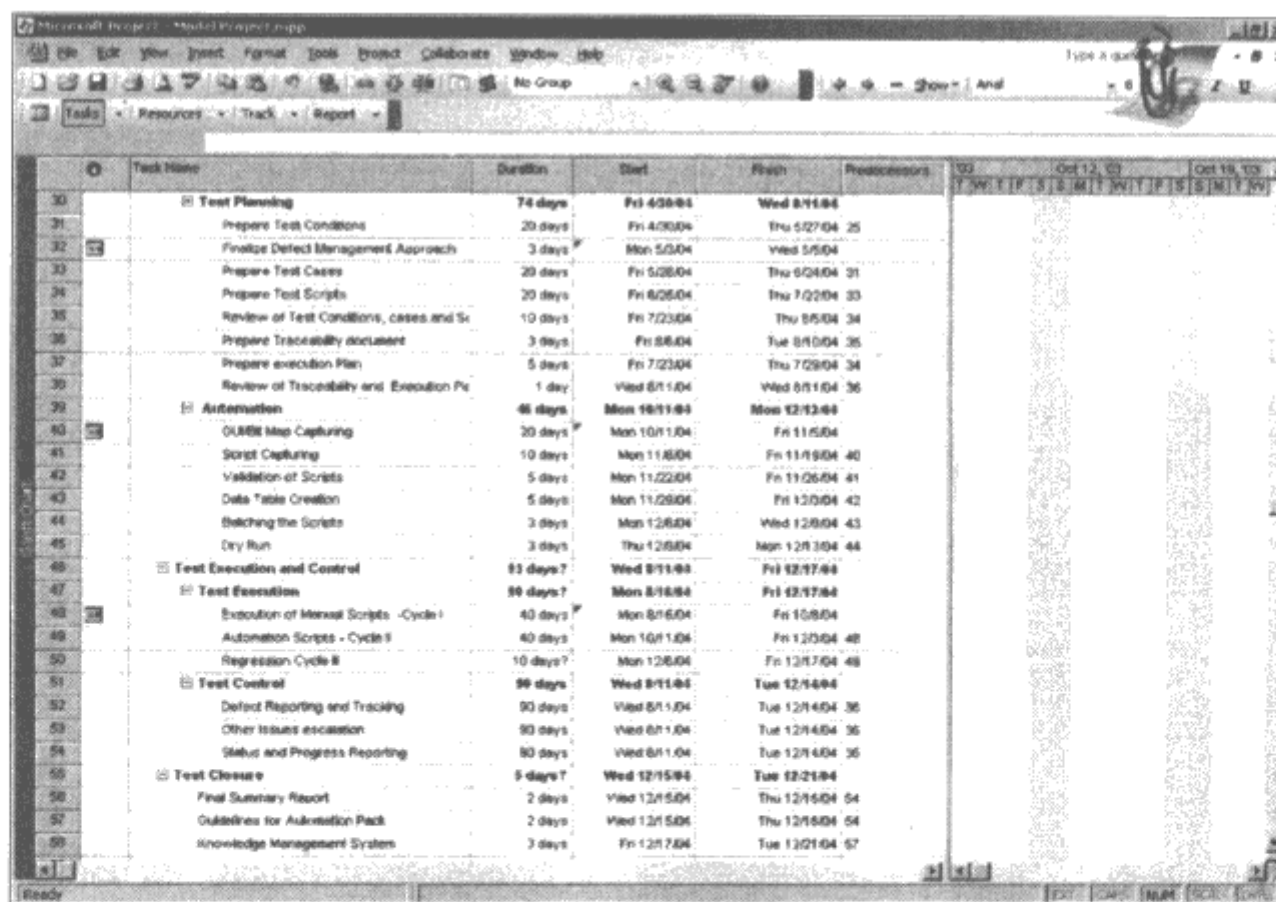


图24-2 测试计划范例（续）

在项目计划的诸多活动中，计划和执行是最为关键的活动，它们决定了测试项目所需资源的

成本和进度安排。在测试的这两个关键环节中估算了许多关键的交付物。这确保了测试团队所用方法更有针对性,也保证了交付物是每个任务的逻辑结果,便于进行项目计划中的下一个任务。但是,要完成某一特定任务并不总是开始进行下一任务的必要条件。项目经理应当分析任务之间的依赖关系。任务依赖关系是两个任务之间的关系,这个关系是一个任务的开始或结束要依赖于另一个任务的开始或结束。依赖于另一个任务的任务是后续任务,它所依赖的任务则是它的前驱任务。

下面描述了一些典型的测试依赖关系,并阐述了为什么它们对于测试管理层来说非常重要。

(1) 结束-开始(FS)。直到任务A结束,任务B才能开始。例如,如果你有两个任务“测试脚本的编写”和“测试执行”,则“测试脚本的编写”完成之前不能开始“测试执行”。这是依赖关系中最常见的形式。

(2) 开始-开始(SS)。在任务A开始之前,任务B不能开始。例如,如果我们有任务“测试脚本的编写”和“运行计划准备”,则“测试脚本的编写”开始之后“运行计划准备”才能开始。

(3) 结束-结束(FF)。在任务A结束之前任务B不能结束。例如,如果你有两个任务“完成测试的执行”和“测试结束报告”,则“完成测试的执行”结束之前,“测试结束报告”是不能结束的。

(4) 开始-结束(SF)。在任务A开始之前任务B不能结束。这种依赖关系类型可以用于“正好及时的日程安排”直到一个里程碑或项目结束日期,以使一个任务完成时间落后的风险降到最低,如果其依赖的任务落后了的话。这种依赖关系类型在需要在一个里程碑或项目结束日期之前完成一个相关任务的时候应用。但是这个确切时间并不真正要紧,我们不希望一个落后于原定完成日期的任务影响到“正好及时的日程安排”。这时,你可以在你希望安排正好及时的日程的任务(前驱任务)和其相关的任务(后续任务)之间创建一个SF从属关系。这样,如果你需要更新后续任务的进度,不会影响到前驱任务的日程安排。

24.6 工作量估算:项目建模

下面描述怎样有效使用一个估算模板。

这一模型中定义了工作量估算的关键活动,包括功能测试。根据定义的参数和来源于项目团队的经验决定这些活动中每一个活动的时间。表24-1显示了项目经理、测试主管和测试工程师通常与哪些任务相关联。

表24-1 工作量估算的活动

测试启动和计划	资 源	
了解应用程序		项目经理、测试主管
培训其他团队成员/歧义性评审	测试工程师	测试主管
项目计划/测试策略		项目经理
测试条件/场景	测试工程师	
测试条件的评审		项目经理
测试用例	测试工程师	
测试脚本	测试工程师	
测试脚本的内部评审		项目经理
覆盖/追溯矩阵的准备	测试工程师	测试主管

(续)

测试启动和计划	资 源	
数据需求/指导原则	测试工程师	测试主管
运行计划的准备		测试主管
运行计划的内部评审		项目经理
业务部门签署批准		
测试执行		
第0天验证——环境检查		项目经理
应用程序测试脚本的确认	测试工程师	测试主管
迭代1 (100%) (执行和缺陷评审)	测试工程师	测试主管
迭代2 (50%) (执行和缺陷评审)	测试工程师	测试主管
迭代3 (50%) (自动化)	测试工程师	项目经理
测试收尾		
最终报告的准备		项目经理
业务部门评审与签署		

在这些脚本的准备时间和执行时间的基础上，测试用例可以分为简单的、中等的和复杂的3类。表24-2估算并输入了项目管理活动和其他项目相关活动所需的基线时间。

表24-2 基线工作量估算

计 划	执 行 ^a
条件到用例	
简单的	1
中等的	3
复杂的	5
缓冲区	20%
用例到脚本	
10	1
每天的测试用例数目	
30	15
每天的测试脚本数目	
2	1
时间线	
小时/天	8
天/周	5
天/月	22
项目进度	
35	25

注：项目基线——可根据项目需求改变的值。

a. 包括缺陷回归。

表24-3分别显示了测试计划、测试执行和测试收尾所需的测试工程师和测试项目经理的总工作量。每个工作量变量都计算了总的人日和总的人月。正常来说，一个月取22个工作日作为一个人月。表中还显示了所需的总人数是可以从人月计算得来。如果测试执行进度表已经在整个里程碑项目计划中定义好了，则在给定时间内完成项目所需的资源数量是可以估算出来的。

表24-3 总工作量和所需人数

编 号	资 源	测试计划/脚本编写	测试执行	测试收尾	总 数
		(以人日为单位的所有工作量)			
1	测试工程师				
2	项目经理/测试主管				
	总的人日				
	总的人月	60.0	30.0	10.0	100.0
	百分比	60.0%	30.0%	10.0%	100.0%
人月（仅测试工程师的工作量）		0	0	0	0
团队规模		4	3	0	7

项目团队应建立关于每个测试人员一天能准备和执行多少测试条件、测试用例和测试脚本的基线。这对于估算是非常关键的，并且会随着项目的不同而有所不同。类似地，对这些活动的每一个而言，评审活动都应被计算为该活动的百分比。

24.7 质量标准

为每个项目做质量管理方法计划包括建立质量标准。这些标准是以客户可以接受的质量级别为基础的。很多公司需要质量陈述，为产品或项目质量定义可度量目标。这种度量需要被审计，以便可追溯到产生该测试度量的测试。本节中描述的这种计划技术和过程将帮助测试团队满足测试标准。

25.1 质量控制与缺陷管理

质量控制过程是项目质量管理的第三个阶段，质量管理的关键要素——缺陷管理建立了记录和组织测试执行阶段发现的缺陷的流程与方法。这一过程的输出为项目利益相关人提供了判断团队执行测试计划时取得的进展的方法。同样的输出也给终端用户提供了一种可视性，关注产品符合其需求的程度。

这一节将缺陷管理过程分解成下面几个基本功能。

- 缺陷发现与分类。
- 缺陷跟踪。
- 缺陷报告。

25.2 缺陷发现与分类

缺陷要么是背离了业务需求，要么是背离了技术需求。测试人员通常在执行测试用例时发现和记录缺陷。尽管测试可以找出缺陷，终端用户使用业务应用或系统时还是可以找到缺陷。

将缺陷分类方便变更管理，而且有助于计划和划分修复该缺陷要做的返工工作的优先级。各个组织的分类方法可能不同。下面是一种分类的例子。

- 崩溃的 (X)：缺陷的影响非常严重，并且由于临时解决方案不可行，所以在问题得到解决之前无法对系统进行测试。
- 严重的 (C)：缺陷的影响非常严重，但是可以执行临时解决方案。在任何情况下缺陷都不会影响测试过程。
- 不严重 (N)：所有不属于X或C类的缺陷都被归结为N类。这些是通过文档和用户培训可以解决的缺陷。这些缺陷可能是图形用户界面方面的，也可能是一些小的字段级的问题。图25-1描述了缺陷的生命周期流。一个缺陷所拥有的最初和最终状态分别是“新缺陷”和“已关闭”。

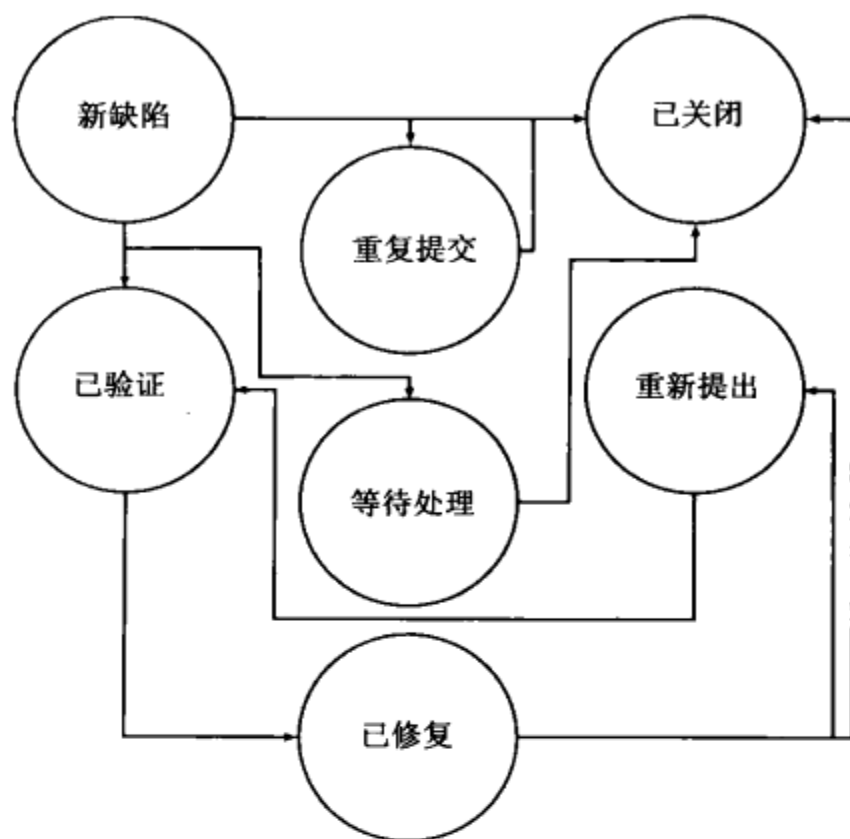


图25-1 缺陷生命周期

25.3 缺陷的优先级

测试活动期间，测试人员在将缺陷记录到缺陷跟踪系统时，为每个缺陷分配一个优先级。因为这个优先级可能会影响开发团队修复该缺陷的顺序，所以分配给缺陷的优先级可能会随着缺陷会议上讨论的结果而发生变化。

下面是最常见的优先级类型。

- 高：只有在缺陷修复后才能进行后续的开发和测试工作。未完成修复工作之前系统是不能使用的。
- 中：优先级为中的缺陷会制约开发和测试活动的进行，因此要尽快解决。在缺陷修复之前，软件系统的使用将受到严重的影响。
- 低：这种缺陷也是需要修复的，但是可以在数量积累到一定程度的时候一并修复。

25.4 缺陷的种类

按照不同的测试策略，缺陷可以分为好多种。下面是在测试项目中经常会看到的主要缺陷种类。

- 测试失误（WAI）：要修改测试用例。当测试人员的理解出现偏差的时候就会出现这种情况。
- 有争议的缺陷：当测试团队和开发团队观点不同的时候就会出现这种情况。这种缺陷将被提交给领域顾问组等候最后的裁决。
- 代码变更：开发团队修复缺陷时会出现这种情况。
- 数据关联：缺陷是由数据造成的而不是编码造成的时会出现这种情况。

- 用户培训：当缺陷不是很严重或者不是在技术上无法修复的时候，就需要培训用户在使用时尽量避免这种情况的发生。理想的情况是缺陷不严重。
- 新需求：讨论后增加新的功能。
- 用户维护：用户对配置和参数的维护导致此类缺陷。
- 图形界面：不包含在以上各类中的其他缺陷，如用户角度的图形用户界面缺陷。

25.5 缺陷跟踪

测试策略文档规定了项目的缺陷管理过程（见图25-2），它清楚地说明了发现需要报告给开发人员和系统拥有者的缺陷时测试工程师应该怎么做。

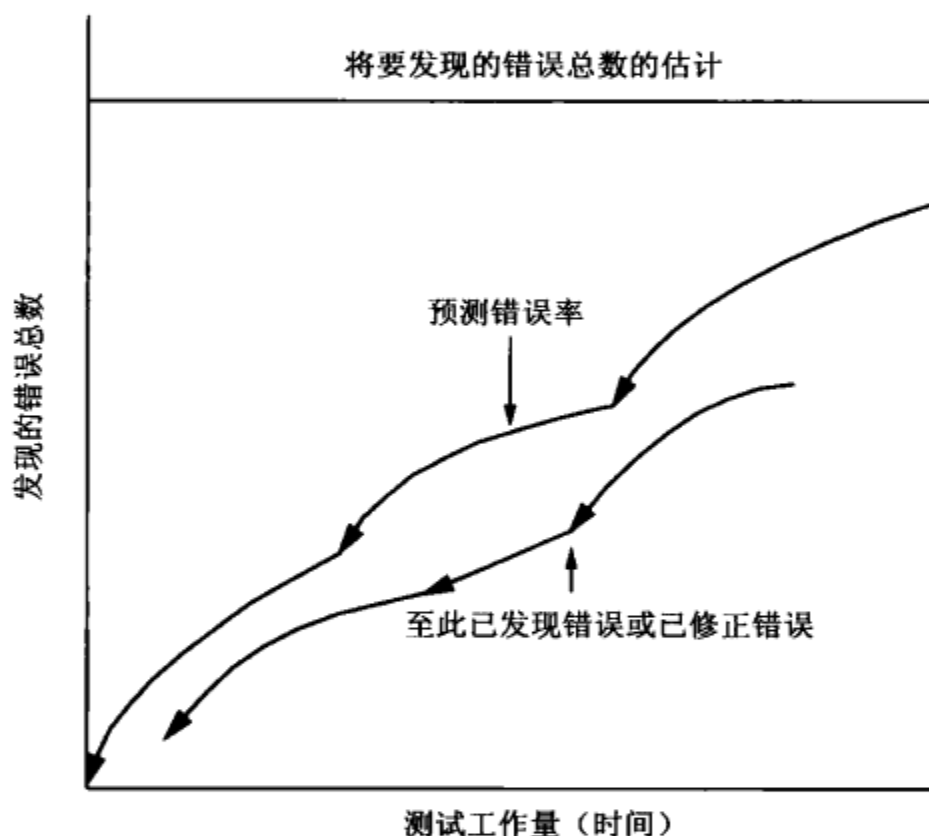


图25-2 缺陷跟踪

测试工程师把发现的缺陷记录到缺陷日志（参见E.9节）中时，应该注明其发现这一缺陷的时间。缺陷日志也可以是包含测试结果以及期望结果和实际结果之间差异描述的数据库。

有很多缺陷管理工具可用于记录和监视缺陷。第六部分会介绍一些流行的缺陷管理工具。

303

25.6 缺陷报告

测试人员利用缺陷报告（也称为问题报告）找到问题的细节，以便能够对问题进行评估，并按优先级将其分到产品缺陷列表中。缺陷报告对被安排修复该缺陷的开发人员以及验证缺陷已被修复的测试人员是非常重要的，对项目管理团队也同样重要。缺陷报告不包括期望测试结果和实际测试结果的详细描述，但需要详细的问题描述。报告缺陷使用E.12节所述的收集信息的标准格式。

25.7 缺陷总结

304

趋势曲线是以缺陷报告中收集的信息为基础的，并且以图的方式展示下面几类趋势。

- 随着时间发现的错误总数。
- 按原因分类的错误，如操作符错误和程序错误。
- 按发现方式分类的错误，如由用户发现的错误。
- 按系统分类的错误，如在订单输入系统中发现的错误。
- 按组织发现的错误，如支持组或操作组发现的错误。

图25-2给出了一个时间与测试期间发现错误数量的关系图。预测错误率是完成测试工作量的进度估计。当修正率成为测试过程的瓶颈时，应该分配额外的开发资源。图25-2还给出了相对于项目错误总量，预测错误率和实际错误率之间的差异。

25.8 缺陷会议

缺陷会议是测试人员、分析人员、开发人员和业务人员之间沟通信息的有效渠道。

每天工作结束后，召开测试团队和开发团队的会议，讨论测试的执行情况和发现的缺陷。正式确定缺陷的种类及严重性就是在这时完成的。

在和开发团队开会前，测试团队应该与测试项目经理内部讨论要报告的缺陷。这一过程可以保证测试团队提交的缺陷是准确、可信的。

25.9 缺陷度量

对缺陷进行分析可以以缺陷的严重性、出现情况及种类为基础。例如，缺陷密度是一种度量指标，指出了在应用系统中特定模块内的缺陷数占缺陷总数的比例。可以基于缺陷管理的各个方面对度量进一步进行分析和推导。

- 缺陷生存期：缺陷生存期是指从发现缺陷到关闭缺陷之间的时间周期。在回归过程中将缺陷集包含在冒烟测试中是客观的做法。
- 缺陷密度：缺陷密度通常按照如下所示的每千行源代码（KSLOC）量来计算。缺陷密度测量方法的使用有助于与期望的缺陷密度进行对比时预测潜在的缺陷数，确定测试是否充足，建立标准缺陷密度数据库。

305

$$Dd = D/KLSOC$$

其中，D为缺陷数，KSLOC为不含注释的源代码行数（以每千行计），Dd为实际缺陷密度。

画出缺陷密度与模块大小之间的关系通常会产生一个凹的U形曲线（见图25-3）。从图形可以看出，非常小和非常大的模块比中等大小的模块的缺陷数要高。小模块的bug发生率的增加已经占据了各种系统，这一点已经被很多研究证明了。

看待同一数据的不同方法是每个模块针对总bug数画一条线，曲线看起来大致是一条对数曲线，然后变平，相当于缺陷密度曲线的最小值，之后以代码行数的平方方向上增长。

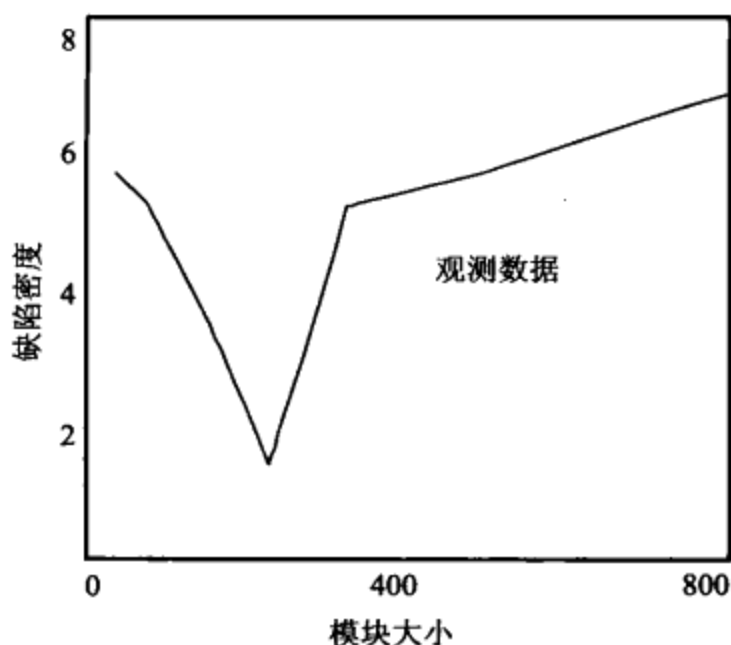


图25-3 缺陷数量及密度与模块大小的关系

25.10 质量标准

管理找出和修正缺陷的周期是质量控制过程中的必须的活动。这项工作的目的是将产品的质量 and 计划的质量标准进行比较。如果项目经理和测试经理还没有很好地建立质量标准，那么质量的成本会达到回报减少的点，在这一点，找出和修正更多缺陷的成本将高于项目的财务收益。

增强质量标准意味着提交客户满意的产品。超出质量验收级别也是回报减少的点，在这一点质量的成本超出项目的财务收益。

26.1 质量控制与测试整合

本节讲述质量控制过程的另一个方面——测试执行。测试的整合方法支持质量控制的目的，这里总结为让开发过程远离错误并且避免错误到达用户手中。

传统上讲，测试组织和开发组织是通过功能特征区分的，这种划分鼓励组织内部不要跨功能线共享资源和知识。对组织效率的期望的增长以及对章程性标准合规性的重视程度的提高，是考虑将开发与测试方法和过程合并在一起的原因。

26.2 测试整合

将开发方法和测试方法整合到一种方法中既不是一个新概念也不是实现常识。无论变更的原因是什么，没有业务案例和行政层面的支持，合并是不会成功的。

将测试和开发方法整合起来的一种方法是，通过增加和修改开发任务将测试任务和步骤整合到开发过程中。通常，由负责开发方法的人来整合测试过程。如果没有人直接负责开发方法，那么测试经理将负责整合这一过程。

另外一种方法是落实质量保证功能的整合部分。行政管理层不必告诉质量保证团队应该采用软件测试标准的哪些方面，也不需要指定应该如何将测试方法整合到现有的设计方法中，或者完成这项任务所需的时间和工作量。行政管理层应该定下基调，声明在组织中整合方法将作为测试的基础。

26.2.1 步骤 1：组建测试团队

创建结合方法从组建完成这项任务的团队开始。要让新的方法能起作用，必须指定既了解测试又了解开发的关键人员来管理这个团队。小组应该由3~7位受同行尊敬的人组成。如果成员少于3人，成功引入测试方法必需的在相互作用和精力方面都不够。如果成员多于7人，团队的管理会非常难。一个有经验的董事长可以和执行发起人一起批准这一团队的任务。项目发起人要确保测试管理团队做到以下几项。

- 理解本书所讨论的测试理念和软件测试标准。
- 定制软件测试标准并将其整合到组织的系统设计方法和维护方法中。

- 鼓励坚持使用并支持整合过的测试方法，同意按照方法所描述的内容进行测试。

26.2.2 步骤 2：确定将要整合的任务

本书第三部分描述了不同的开发方法和测试方法，反映了业务技术、系统架构和组织结构之间的关系。要确定是否整合设计方法，也必须评估同样的关系。团队完成这一步的时候，成员必须就测试的一般目标以及设计方法如何影响测试方法达成一致意见。设计方法可以用设计标准来表述，以便团队可以接受设计任务作为整合方法的一部分，或者决定设计方法和任务仍然放在整合方法之外。

310

26.2.3 步骤 3：定制测试步骤及任务

团队应该定制本书中介绍的步骤和任务，以便与企业的开发和测试方法一致。团队可以亲自完成这个定制任务，也可以安排给他人完成（如负责设计方法的团队）。

定制通常包括以下两个方面。

- 标准化词汇表——词汇表在整个设计方法中要一致。如果全体成员都理解并使用相同的词汇表，就可以很方便地进行工作交接。定制词汇表意味着改变测试中的标准词汇表，或者将测试使用的词汇表与系统开发方法的词汇表进行整合。
- 改变表述结构——本书所描述的测试步骤与任务可能会与设计方法的其他部分有所不同。例如，本书将软件测试工具的形式和内容描述作为单独的章节。如果将其与系统开发方法合成一个单元，这些内容可能就需要重新整理或排列，使其与开发手册保持一致。

26

在测试计划阶段，测试团队会确定在正在开发的系统中使用哪些测试标准、规程、任务、工作表及检查表。团队应该为单独的应用系统或为特殊的开发功能定制过程。

团队还可以为了确认这一过程是否按期望的方式工作，选择过程更小的版本。

26.2.4 步骤 4：选择整合时间点

这一步骤包含的内容为，在开发方法中选择一些时间点与测试步骤和任务进行整合。这要求对于系统开发方法与软件任务有透彻的理解。确定整合时间点有两个重要标准。

- 所需数据——只有在整合时间点所需的信息生成后，才能将测试任务插入设计方法中。
- 需要测试结果的地方——测试任务必须在系统开发方法需要测试结果之前完成。

311

遵循以上两条准则就可以确定执行测试任务的最早时间点和最晚时间点。测试任务应该在有些时间点插入到开发方法中。

26.2.5 步骤 5：修改开发方法

在这一步，修改系统开发方法所需的所有信息都是可用的。这一步需要一个熟悉设计过程的人来将测试过程和步骤插入到开发方法文档中。

26.2.6 步骤 6：对测试方法进行培训

这一步骤包括对分析人员、用户和程序员进行测试方法培训。一旦测试整合到系统开发方法

中，所有人员就必须接受培训并促使其使用这种测试方法，这是更加困难的工作。

测试管理团队在说服其同事们接受并使用新方法的过程中扮演着重要的角色。首先，以身作则；其次，积极鼓励大家采用新方法。这一步的重要部分是组织测试研讨会，包含下面的内容。

- 测试概念及方法——关于这部分的培训内容包括在附录F中会再次提及。
- 测试标准——负责测试的人必须知道测试的衡量标准。首先要教会测试团队成员这些标准，这样他们才能知道为什么要完成某些特定的任务（如测试步骤），然后是对测试规程进行培训。如果他们认为完成这个规程只是测试的一种方式，他们会采用更好的方法。另一方面，如果参加测试的人员能够了解执行这个测试规程的目标（例如，是为了达成测试标准），那么大家将更乐于学习并遵循这个规程。
- 测试方法——测试与系统开发方法相结合的方法在推广时要循序渐进。分析人员、用户和程序员要从一开始就在讲师的指导下完成任务。这样有助于保证这些专业人员能够充分理解测试任务如何执行并知道所期望的结果。

312

在测试负责人经过培训并熟练掌握测试规程之前，管理层应该允许某些测试错误。此外，在他们没有熟练掌握测试规程之前，在规程执行期间要接受严格地监督。

下面将介绍当测试过程整合到开发方法时，如何进行缺陷记录和分析规程。这一规程需要对缺陷进行分类，并确保缺陷在整个开发方法中被适当记录下来。

26.2.7 步骤 7：合并缺陷记录

测试人员工作的一个组成部分是质量控制活动，必须记录及分析缺陷，以确定如何改善整合过程。这个过程是与操作应用系统中报告问题等价的。测试经理必须要能够捕获关于出现的问题和缺陷的信息，没有这些信息很难改善测试。

缺陷记录最困难的部分是说服开发人员相信这些信息不是针对他们个人的。要严格地收集这些信息用以改善测试过程，但是这些信息决不能用于绩效或其他对个人的评估。

26.3 整合后的团队

虽然项目经理和测试经理都是关注项目在项目的预算和进度约束条件下完成的人，但执行阶段的结果往往掌握在开发人员和测试人员的手中。整合后的团队方法并没有改变这一点。如果为了一个正当的理由完成整合，并且时刻记得质量是最重要的，那么即将出现的改变，就是开发人员理解测试人员如何思考以及测试人员更了解开发人员思想的机会。

313

汇聚我们的技术知识，最大的挑战——理解客户的视角——仍然没变。

27.1 组织架构

在项目框架的上下文中，项目经理的作用是确定并调整组织架构，以实现项目的目标。“组织架构的约束”部分描述了质量保证组织的结构组成与项目经理在保证项目质量时所负的责任之间的关系。

描述所有的组织架构几乎是不可能的。本节主要介绍使用两种不同的组织架构的项目。

- 在过程驱动质量方法和交付过程已经建立的情况下，保证项目质量。
- 在质量基础架构不存在的情况下，保证项目质量。

27.2 已建立得很好的质量保证部门的特征

一个已建立得很好的质量保证部门具备以下特征：

- 与业务单元整合（策略上和技术上）；
- 对产品和服务的提交过程进行可度量的、有针对性的改进；
- 当成品通过产品生命周期移交时，减少客户对已交付产品和服务的质疑。
- 产品和服务的组合要经得起检验，保证较高的客户满意度。

图27-1展示了一个组织架构的示例，这个组织架构将质量保证部门和项目经理整合到业务机构中。注意，这个组织架构是如何将首席质量官与首席信息官放在同一层的。这种结构将质量保证组和质量控制组置于企业的交付结算期。虽然质量的成本是可以测量的，但这些组很少会被组织为一个业务单元。

27.3 职责划分

维护项目质量和交付优质产品都是项目经理的职责，但具体的质量管理任务却是项目经理和质量经理共同的职责。质量经理负责管理测试的准备、执行并编写测试报告。

与质量有关的项目职责包括以下几个方面。

- 报告项目状态——告知测试工作是否按计划进行。
- 评估项目状态——分析测试结果，预测测试工作是否按计划进行。
- 告知变更——项目变更管理活动包括项目范围、进度安排、成本以及质量。

- ❑ 缺陷跟踪和评审——确认那些必须执行的修改工作已经用日志记录、分配、完成并得到了确认。
- ❑ 确保资源物尽其用——根据项目的利润，增加或减少项目。
- ❑ 不断进行过程改进——从分析质量管理过程的输入和输出中获取相关知识并加以应用。分析工作包括评估工具的使用效率，理解改进测试技术和测试过程的方法，评估技术培训和测试环境是否一致。
- ❑ 确保质量保证部门能调整生产者的视角与消费者的视角一致。

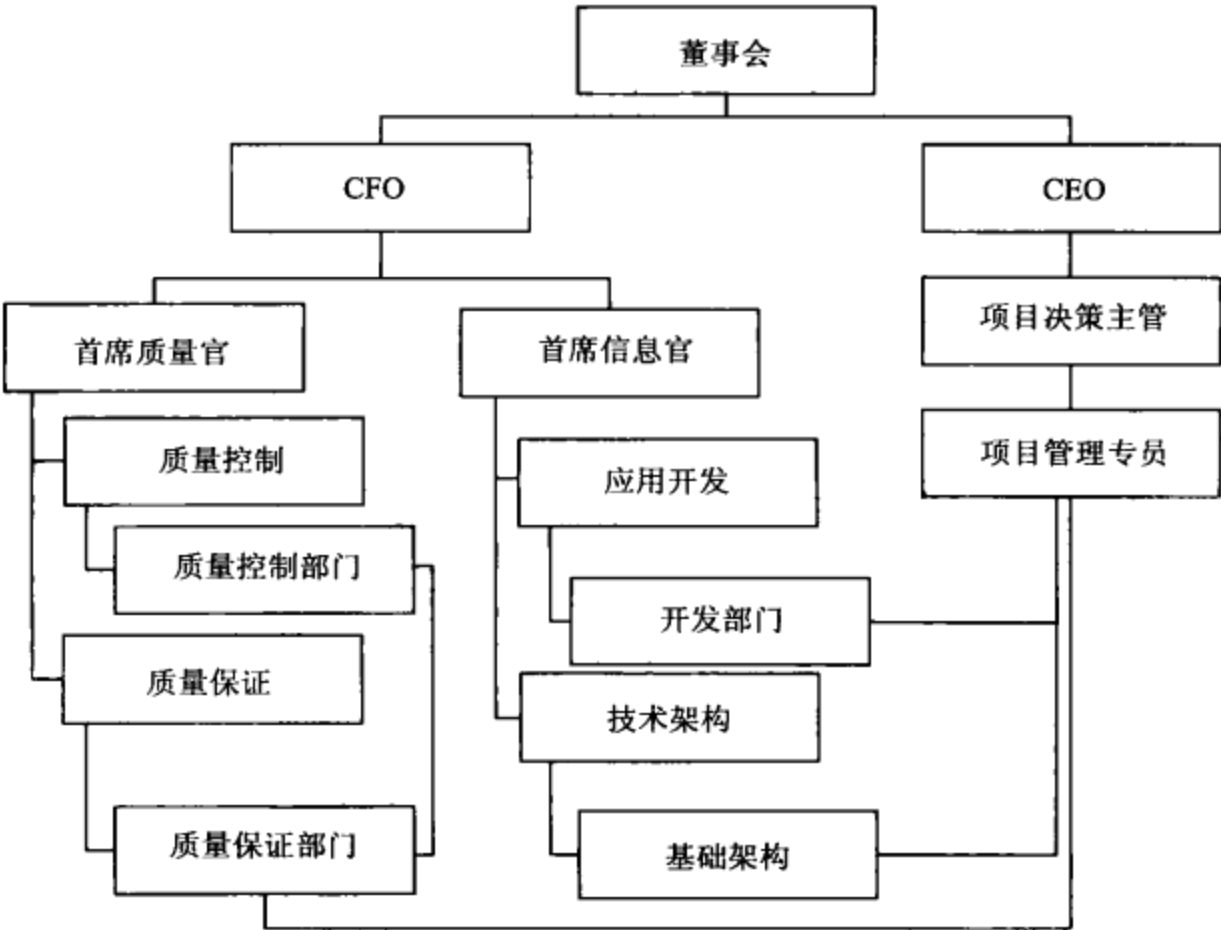


图27-1 矫正质量、开发和项目管理

27.4 组织关系

项目经理与质量保证部门的关系既可以增加也可以降低项目成功的几率。项目经理和质量保证及质量控制团队的交流方式经常被人忽略。

表27-1总结了影响项目经理和质量保证部门之间关系的正确观点和错误观点。

表27-1 正确观点和错误的观点

正确的观点	错误的观点
项目经理在项目开始和工作评估的第一时间介入质量团队	项目经理认为产品质量过程似乎会影响项目进度和成本约束
项目经理与项目发起人就质量资源的最优使用进行协商	项目经理不了解质量团队在支持项目的过程中所起的作用

(续)

正确的观点	错误的观点
项目经理将质量度量集成到项目性能测量中	项目的管理过程是多余的,并且会给质量过程增加不必要的复杂性
项目经理支持质量控制团队关于质量产品收益的决定性协商的调查结果	项目经理做出的决定对产品质量产生了不利影响,但把责任推给质量组

27.5 在质量基础设施不存在的情况下使用项目框架

项目经理遇到一个没有正式质量基础设施存在的组织架构时,通常会进行即兴测试,虽然与正式测试组合时,即兴测试(探索性测试)是很高效的方法,但这里所介绍的即兴测试是无文档、无计划的,是专用的一种测试方法。

下面这些组织行为描述了即兴测试的特征。

- 允许在准生产环境中加入新功能和bug修正,以判断这些修改是否会在现存代码中引起预料之外的结果。
- 当开发人员将包含新功能(包括bug修正)的产品加入到生产环境中时,最终用户验证产品是否满足他们的需求。
- 产品支持工作量大于等于开发工作量。如果追踪一下,支持成本和开发成本之间也存在这种关系。
- 待处理的bug修正和增强方面的请求使产品工程师应接不暇。
- 在产品和服务整合中,没有一致的标准来测量产品和服务的质量。

27.6 即兴测试和项目框架

项目收益的即兴测试的关键是,利用项目框架注重“需求和测试工作量之间的可追溯性”这一特点。

在前面几节中,已经建立了项目范围和产品范围之间的联系,方法是假设产品范围描述产品(或即将交付的服务)的特征,项目范围指定交付产品所需的工作。实际上,项目经理利用即兴测试活动来统一生产者和客户对质量所持的不同观点。

27.7 使用可追溯性/确认矩阵

产品经理使用一个简单的矩阵追踪即兴测试过程,并评估产品质量等级。这个矩阵将需求可追溯性和功能确认组合起来,如表27-2所示。

表27-2 可追溯性/确认矩阵

用户需求参考		技术需求	通过/失败	确 认 者	日 期
客户登记	1.1客户必须是有效的	1.1.2在线客户界面			

这个矩阵格式将业务需求和技术(系统)需求组织到功能区,由最终用户进行确认。当最终

用户确认功能需求满足预期时，他就在矩阵里标明验收通过。该矩阵还给出了没有通过验收的功能，但没有表明该功能何时重新进行测试。项目经理遵循项目变更控制过程来管理和报告测试与重新测试的进展。

可追溯性/确认矩阵依赖于书面的功能需求和技术需求的有效性。如果没有正式的需求文档，项目经理和他的团队将使用相关制品编写基本需求。制品获取用户需求和生产者需求的说明。创建需求的制品包括功能的书面请求、业务流程图、过程模型、UML图和设计文档。

27.8 进度报告

确认工作开始以后，项目经理负责与项目发起人和利益相关者交流测试进度。项目经理的测试总结报告是分析需求覆盖率和追踪满足最终用户需求的功能之后的结果。该报告获取了需求和用户验收之间的从属关系，如表27-3所示。

表27-3 按功能区做的测试总结报告

功 能 区	用 户 需 求	已 验 收		剩 余	
		计 数	%	计 数	%
客户登记	16	9	56.25	7	43.75
客户报告	26	25	96.15	1	3.85
商家登记	19	5	26.32	14	73.68
商家报告	26	14	53.85	12	46.15
总计	87	53	60.92	34	39.08

即兴测试暗指测试工作不在计划内，但测试的结束很少是在任意时间点的。项目经理使用前面提到的报告来评估测试对项目进度的影响。这种影响是通过项目的计划完成时间和项目的确认工作结束时间之间的差值来衡量的。

即使项目的计划完成时间应该根据计划外的工作做相应调整，但实际上，项目进度却只能扩展到这个程度。创建可追溯性/确认矩阵和测试总结报告的好处在于，这样一来，产品和项目的质量变成了实际可测量的东西，从而可以指导项目发起人的业务决策。

319
320

软件测试中的新兴专业领域

软件开发是一种持续变化、不断创新、竞争日趋激烈的行业。因此，紧随软件测试的发展趋势很难。本部分将描述软件测试中的新兴专业领域，这些领域能够帮助测试团队改进被测试的应用的质量和性能。

我们（以及一些实践经验丰富的专家）介绍每一主题的目的是展示软件测试的新趋势，并重点介绍一些有前途的观点。著名论坛和会议上都会有相关文章出现。

本部分的目标如下：

- 描述如何执行软件测试过程评估；
- 提供评估和启动软件自动化项目的方法；
- 描述各种测试自动化框架的特征；
- 描述各种非功能测试类型，包括性能测试、安全性测试、易用性测试和合规性（compliance）；
- 定义SOA测试的主要步骤；
- 概述在敏捷开发环境中如何执行软件测试；
- 列出启用软件管理的基本步骤，以确保为审计文档做好准备；
- 描述COE（center of excellence，卓越中心）的组织结构；
- 描述如何建立COE；
- 描述了本土人事方法和离岸人事方法的优缺点。

321

322

在IT方面进行投入的公司进行测试过程和自动化评估的目的是增强竞争力，降低开销，符合法规需求。业务新方案中详细介绍了这些业务目的，但几乎都不能度量变更对现有IT基础架构和IT业务系统所造成的影响。

相反，IT软件开发和实现通常会忽略企业的正当理由，而是将重点放在技术方案上（更详细的介绍参见第30章）。

本章将介绍管理业务环境中测试过程评估的方法。

28.1 测试过程评估

我们通常会认为，软件测试并不会给企业创造直接的价值。但是，经过一段时间之后，很多企业都意识到软件测试是必不可少的，借助软件测试，可以避免那些对企业本身产生不利影响的灾难性的错误。

软件测试过程不但检测产品是否符合设计要求，而且还验证产品是否实现了业务目标。如果质量保证和质量控制与业务目标不一致，那么预期的业务就会受到功能缺陷（或bug）的威胁。

323

只注重IT系统设计而忽视功能性业务测试通常会导致生产环境下业务应用不可用。

当一个关键的业务系统失败时，很多大中型公司实施错误纠正策略，借此估计企业的经济损失并找出导致失败的根本原因。通常会从质量保证开始，一路追踪错误，直至追踪到未经测试的代码，回归测试中不包括这些未经测试的代码。

Y2K恐慌使很多公司意识到将测试过程整合到软件开发生命周期中是多么重要。实际的业务现状促使软件测试过程评估受到关注。

当前软件工程和管理方法的评估就是一个良好的起点。差距分析报告中详细介绍了这方面的研究成果，该报告通过以下几个方面分析了该公司软件测试方法的优缺点：

- 管理层的讨论；
- 问卷调查；
- 反馈；
- 精心组织的采访；
- 行动方案。

分析工作包括分析各个测试阶段所需的应用和测试输出结果。接下来是分析详细的行动计划

和客户的管理，从而找出改进软件测试过程的方法。差异分析活动的重点是，分析那些能改进测试过程的关键区域。

要完成综合分析，过程分析人员不仅要了解业务的运作过程，还要了解每一级别的管理工作对分析的预期都是独一无二的。

28.2 过程评估方法

图28-1给出了测试评估过程的步骤。



图28-1 过程评估方法

28.2.1 步骤 1：标识关键元素

过程分析人员评估用于开发应用程序的技术，研究客户采用的软件开发方法，判定公司的成熟水平，以及检查现有测试过程的当前水平。 [324]

测试过程评估确定高质量应用和产品的成熟度和覆盖率。

下面是在测试过程评估过程中研究的内容：

- 测试方法的范围；
- 测试过程的管理；
- 测试功能和培训；
- 生命周期评审方法；
- 评估和计划测试周期；
- 测试策略；
- 测试覆盖率；
- 测试设计技术；
- 测试度量；
- 测试数据；
- 测试组件管理；
- 测试工具；
- 测试环境；
- 缺陷管理；
- 变更和配置管理；
- 交流和报告；
- 委托和动机；
- 静态测试技术。

上面介绍的这些元素比较全面，并不是每个过程改进都会用到所有这些元素。

28.2.2 步骤 2：收集和分析信息

过程分析通过从以下业务代表中收集相关信息，来评估当前的测试过程（如图28-2所示）。

- 业务单元主管；
- 项目经理；
- 程序经理；
- 质量保证经理；
- 产品经理；
- 测试工程师；
- 测试组长；
- SQA组长；
- 测试工具专家；
- 测试环境专家。

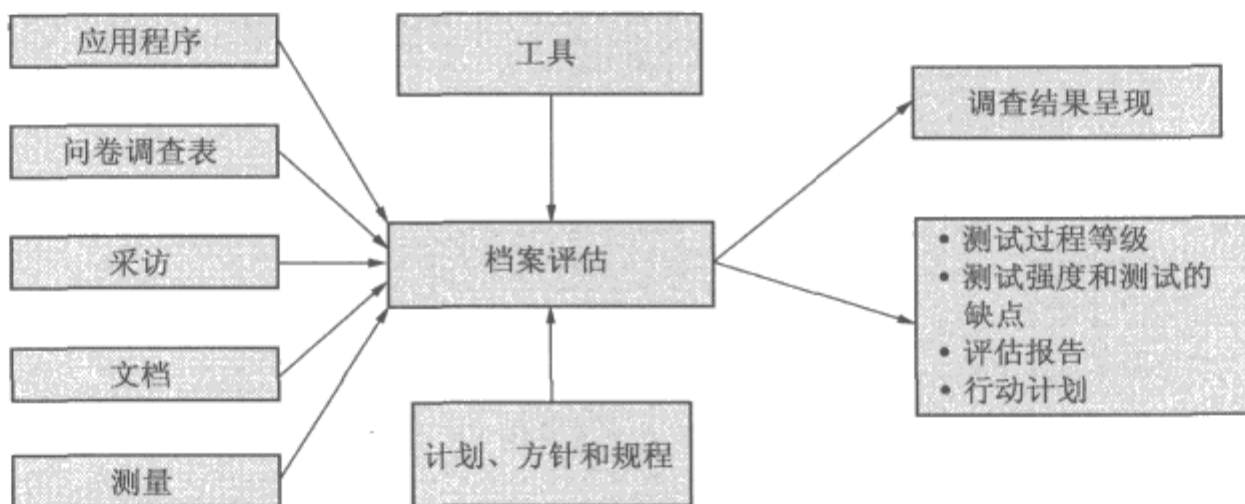


图28-2 测试评估输入和输出

以业务代表的作用和职责为基础，为每个业务代表创建一个问卷调查表，以此来收集信息。进行面谈，并针对现有开发文档和测试文档验证结果，以此来检测需要进一步澄清的内容。

28.2.3 步骤 3：分析测试成熟度

验证并整理了信息之后，过程分析人员就能确定当前过程和为技术组和业务组定义的标准过程集合之间的差异：

- IEEE 829测试文档；
- ANSI；
- 萨班斯-奥克斯利法案（COBIT的ISACA子集）；
- SEI-CMM（Software Engineering Institute-Capability Maturity Model，软件工程研究所能力成熟度模型）。

下面是测试过程的关键部分，这些指导将帮助评估人员总结成熟度。

1. 需求定义的成熟度

成功的测试的基础是，需求是可测试的。在软件开发生命周期的早期阶段开发的需求几乎都是不完整、不明确的。需求的来源迥然不同，但常见来源包括电子邮件、文字描述、未记录的客户需求预期和“小道消息”。最终，需求定义过程必须从各种形式的不明确的信息中提炼出意思简单明了的说明，测试工程师可凭借这些说明来开发测试用例。

326

过程分析人员必须检查现有的需求定义和验证/确认过程。要确定需求分析过程的质量，需要以下几个步骤：

- 收集或推导；
- 分析并按优先级排序；
- 记录；
- 评审完整性；
- 将修改整合到基线文档中。

分析人员还必须评估是否针对需求的变更（包括范围潜变）进行了足够的影响分析，并评估如何为测试过程的每个组成部分管理这些变更。

下面的组织评估是现有需求过程和最佳实践之间差异分析的重要来源。

- 分析开发团队与业务团队的电子邮件交流。
- 分析项目文档以查明需求到测试用例的可追踪性。
- 由企业负责人和质量保证团队核对经过评审的最终需求文档，并签字确认。

观察了现有的需求方法和需求成熟度以后，过程分析人员应该推荐解决差异、改进现有需求定义和验证/确认过程的操作。

2. 测试策略的成熟度

当计划不周导致测试生命周期中出现多个问题时，就用到了测试策略，如测试资源和测试环境的可用性。

测试策略应该是全面的，包括以下几个部分：

- 测试的范围；
- 测试的类型；
- 可追溯性方法；
- 工作量评估；
- 测试用例准备；
- 测试执行方法；
- 缺陷管理过程；
- 资源分配；
- 测试结束过程。

过程分析人员应该核实测试策略是否定义了测试进入标准（为开始测试所做的准备）和测试退出标准（用于完成并停止测试），以及在测试的各个阶段收集的度量类型。

过程分析人员还应该核实测试过程审计度量是否符合质量标准和持续过程改进。测试工具将显示捕获的成本、质量和进度度量。

28

327

过程确认的其他重要内容如下：

- 正式的配置管理过程；
- 用文档记录的端到端的测试过程，以及所有关键测试流程区域的测试规程（包括指导方针、模板和检查表）；
- 经过测量的缺陷流转时间，也就是根据严重程度改正缺陷所花费的时间。

3. 测试工作量估算的成熟度

评价测试工作量的估算方法。不准确的测试估算不仅会延迟测试周期，还会严重影响整个项目进度。

测试工作量的计算采用估算方法来完成，如SMC（Simple, Medium, and Complex test case）模型、WBS（Work Breakdown Structure）模型、测试用例点模型和其他形式的功能点分析。

无论使用什么样的模型，分析人员都应该核实文档记录的估算方法以及将估算工作量与实际测试工作量进行比较的估算方法，以核实估算的准确性水平。

过程分析人员都应该考虑以下几个问题。

- (1) 简单、中等、复杂测试用例的定义。
- (2) 测试数据和用于生产数据的技术可用性。
- (3) 缺陷管理过程需要的工作量。
- (4) 发布前要考虑的测试交互的数量以及回归测试和重新测试采用的方法。

4. 测试设计和执行成熟度

测试设计方法定义了如何设计测试用例，如何建立可追溯性矩阵，以及如何将测试数据链接到测试用例。要优化测试的执行，可以通过差距分析显示在何处缺少过程和链接，导致了无法有效地执行测试。

下面是成熟的测试设计和执行过程的参数。

- 测试是经过测量和量化的过程。
- 测试产品的质量，如可靠性、易用性和可维护性。
- 收集测试用例并记录下来，以备复用。
- 记录缺陷，确定缺陷的严重性。
- 定义和管理测试。
- 度量测试成本和测试效率。
- 测试过程经过很好的调整并不断改进。
- 实施缺陷预防和质量控制。
- 自动化测试在质量控制过程中具有重要意义。
- 工具支持测试用例设计和度量收集。
- 采用过程复用。
- 采用引起缺陷的根本原因分析，也就是缺陷预防技术。

测试过程评估定义了上述所有指标，并收集测试执行度量，以量化持续不断的质量改进。

5. 回归测试的成熟度

必须定义回归测试策略。回归测试可选择重新测试系统或重新测试组件，以核实所做的修改

没有引起意想不到的结果，并且系统或组件仍然符合其规定的要求（IEEE，1900）。测试人员必须确定回归测试的度，使风险最小化。软件的变更可能对看似与软件无关的部分产生意想不到的影响。

6. 测试自动化成熟度

在为缩短测试生命周期和降低测试成本已做了很多工作的这种情况下，测试自动化作为手工测试的替代品应运而生，降低了测试生命周期的成本。但是，在测试工具和脚本开发工作上的最初投入成本仍然十分巨大。企业并不会很快认识到ROI（Return On Investment，投资回报率）。

测试自动化的结构化方法可以通过对开发的代码进行测试，以便软件测试就不会消耗软件生命周期的大部分时间，确保企业获得全部收益。没有计划的测试方法致使很多公司在各种形式的测试中花费达到软件开发生命周期成本的30%之多。

测试策略确定了自动化的范围、要进行自动化的功能、方法和实现自动化的方法。测试策略定义了各种角色和职责、项目测试进度、测试计划、设计活动、测试环境准备、测试风险和意外以及整体的可接受水平。测试策略包括测试规程、命名规范、测试规程格式标准和测试规程可追溯性矩阵。

329

下面是测试自动化策略的标准提纲，可以根据测试需求进行定制（参加E.30节）：

- 项目概览；
- 自动化目的和目标；
- 自动化的范围——包括的和不包括的；
- 自动化方法；
- 测试环境；
- 使用的工具——脚本和测试管理；
- 脚本命名规范；
- 资源和日程安排；
- 培训需求；
- 风险和降低风险的方法；
- 假设和约束；
- 进入标准和退出标准；
- 验收标准；
- 可交付物。

28

28.2.4 步骤 4：记录和呈现结果

最终的差距报告、测试过程结果是关键的可交付物，它们标识了可能需要进行的测试过程改进。确定差距时，差距报告还记录了当前环境中存在的最佳实践。差距报告也可作为当前进程的基线以及将来继续改进方案的出发点。

28.3 测试自动化评估

测试自动化方法决定了如何确保业务需求和应用程序最终目标的实现。该方法帮助计划和确定使用测试自动化进行测试的软件组件。它也确定了针对不同的项目生命周期进行自动化测试的

上下文和方法。

最佳的自动化测试策略必须在缺陷的成本/风险和大量测试的总成本之间取得平衡。目标是完成尽可能多的测试，最小化测试工作量和测试周期，以达到一个可接受的风险等级。

330

下面是测试自动化时应考虑的主要因素。

- (1) 标识正确的应用以及这些应用可被自动化的合理的百分比。
- (2) 标识可使用的测试工具，包括兼容性、成本、易用性、可复用性、框架因素和培训。
- (3) 标识并创建测试框架，如以数据为中心的方法、以业务函数为中心的方法和两者混合的方法。
- (4) 标识可复用测试组件的不同等级，也就是被测试的应用程序中可以复用的函数。
- (5) 创建遵循标准和指导方针的测试自动化脚本。
- (6) 需求确认、检查点、错误处理机制和结果报告。
- (7) 为运行脚本创建相关的测试数据。
- (8) 验证脚本，以确保它们执行了所需的业务函数确认，这些确认是企业所期望的。
- (9) 为维护脚本创建必需的文档，这些脚本是针对功能增强、新发布版本、工具指导手册等开发的。

图28-3所示为PDCA（Plan-Do-Check-Act）模型上下文中的测试自动化方法（参见第一部分的第5章）。与其他持续质量改进方案一样，必须计划、执行、检查测试自动化工作，以核实测试自动化工作正常进行，并起到调整计划的作用。

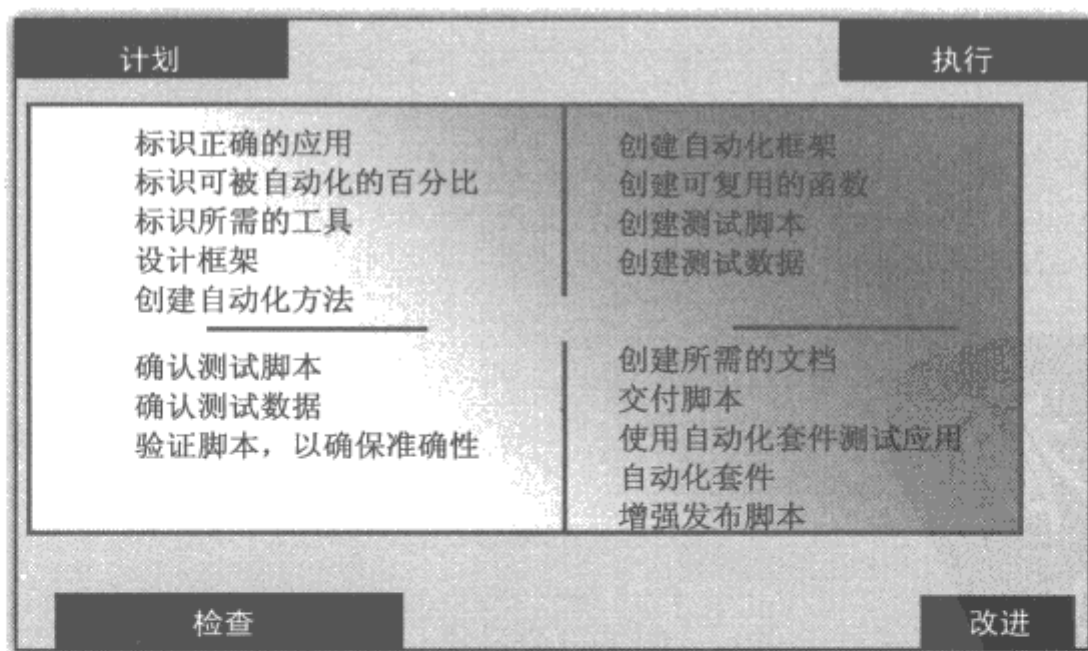


图28-3 应用到测试自动化的PDCA

331

接下来的几节介绍了计划测试自动化策略时应考虑的因素。

28.3.1 标识需要自动化的应用

有些公司迫切希望迅速完成结构测试，所以购买了各种测试工具，但经过一段时间以后，他

们通常都无法从投入中获得预期的收益。失败的主要原因是自动化活动无计划、不系统。

下面是需要评估的主要决定因素，以正确地标识要测试自动化的应用。

- 对业务至关重要的、使用频率高的、具有较长生命周期的应用。
- 对多个平台的应用进行本地化/全局化。
- 每次都需要完整的回归测试的多个发布。
- 需要手动干预的最低限度的外部接口。
- 其发布不会对整个回归测试工作产生不良影响的应用。
- 涉及位图的基于图形用户界面的应用。
- 在所有应用中，其对象和函数被多次使用的应用。
- 前端图形用户界面和后端图形用户界面无需进行太多修改的稳定的应用。

28.3.2 确定最佳测试自动化工具

工具评估对于自动化测试至关重要。测试人员需要收集用于开发应用的技术细节、与应用交互的技术细节、用于开发应用的第三方工具。

每个测试自动化工具的厂商都标明了工具支持的技术类型，以及要进行其他交互和使用底层技术所需的附加设备。在了解产品的基础上，应对各种测试自动化工具进行评估。

在选择测试自动化工具之前，应考虑下面这些常见的因素。

- 进行自动化的应用的技术能力。
- 与应用环境、组件和接口的兼容性。
- 易于测试开发。
- 测试的可维护性。
- 可靠性和良好的市场表现力。
- 使用的自定义对象，使用的第三方工具。
- 厂商提供的工具用法说明。
- 测试人员容易使用。
- 使用的脚本语言，该脚本语言易于使用。
- 脚本易于修改。
- 自动化工具的成本和每年的维护成本。
- 厂商对新技术的支持，以及遇到问题时的技术支持（参见第六部分，了解选择自动化工具的正式方法和非正式方法）。

28.3.3 编写测试脚本的方法

下面的这些活动通常会在测试自动化脚本的过程中涉及，是自动化策略中应该考虑的因素。

- 测试用例选择：评审所有测试用例，适当地与相关业务范围比较，这样就可以减少几个测试用例。将可被自动化的测试用例和不可被自动化的测试用例分开。
- 捕获基础流程脚本：捕获基本业务流程、图形用户界面或位图，遵循脚本的编写标准和指导方针，并使用可用的库函数。

- 验证和确认：重新调整脚本，添加所需的检查点、断点、函数和同步机制。
- 创建数据表：创建所有可能的测试数据组合，以确保测试数据的覆盖率，并为可追溯性做准备。
- 验证脚本：运行脚本，确认结果并遵循缺陷管理流程。

28.3.4 测试执行的方法

通常，测试管理工具（如惠普的Quality Center）用于存储所创建的自动化脚本。使用测试管理工具中的可用功能触发这些脚本以便执行。

自动化脚本的优点是执行时无需人为干预。这些脚本可安排在环境可用时（甚至是在午夜）被触发。当自动化分析人员第二天回来时，测试执行的结果已被存储在定义好的文件中了。这些结果可以帮助他们分析和引发异常。

按照已定义的缺陷管理流程，每个测试团队都要完成问题报告。自动化的缺陷追踪工具可为撰写文档和记录软件问题报告提供极大的方便。功能测试和集成测试采用的缺陷管理流程同样也适用于在自动化测试脚本的测试执行过程中发现的缺陷。

333

测试团队经理负责确保测试是按照进度执行的。分配测试人员，测试工作中出现问题时测试人员会在必要时被重新分配，以处理这些问题。要有效地执行这项易失察的功能，测试经理需要进行测试程序状态跟踪和测试程序状态管理报告。

测试度量为测试经理提供了测试覆盖率、测试进度以及测试工作质量的关键指标。度量收集的重点是测试的广度，以包括示例功能的数量和执行测试的数量。（有关测试度量的更多信息参见第22章。）

28.3.5 测试脚本维护

测试团队应该按照测试的执行来评审测试脚本的性能，以判断何处能实施改进，以便下一次迭代时改进测试脚本。在测试执行结果、业务流程中的修改和基本功能增强的基础上升级测试脚本。

无论应用中何时引入新的增强，测试经理都需要对所有新功能或经过修改的功能执行影响分析，分析这些功能如何影响现有回归测试集，以及新脚本如何添加到回归测试集中。

在整个测试执行周期中，测试团队都需要收集各种测试度量。测试评审的重点包括评估应用是否符合验收标准、应用是否可作为产品生产。评审内容还包括评估已完成进度的度量和其他收集到的度量。

在整个测试生命周期中，记录并开始评估在每个里程碑阶段学到的知识的做法很不错。在整个测试生命周期中（尤其是在测试执行阶段）收集到的度量，有助于定位需要解决的问题。

在整个测试过程中学到的度量评估、相应的改进活动和正确的操作都要记录在便于查阅的中心资料库中。

28.4 测试自动化框架

软件测试专家采用了测试自动化技术，因为它是改进质量、降低成本、缩短测试周期行之有

效的方法。很多公司都使用这些测试工具，希望能优化测试工作和质量。但是，经过一段时间以后他们发现，这些工具的作用远没有预期中的那样强大。分析失败的根本原因时他们发现，这是缺少结构化的测试自动化方法以及该方法的整体框架所致。这就引入了各种测试自动化框架，具体框架取决于不同的应用技术，以及测试相关应用时所采用的不同方法。

下面是测试自动化领域中一些比较流行的测试自动化框架：

- 数据驱动框架；
- 模块化框架；
- 关键字驱动框架；
- 混合框架。

本节简要介绍上述框架的各种特征以及构建这些框架的方法。

自动化框架作为一种概念出现，它是规则、假设、标准和指导方针以及常见的可复用组件和为自动化测试软件提供支持的实践方法的集合。自动化框架也定义了有效地使用和维护自动化脚本的目录存储结构，并定义了记录和发布测试自动化结果的方法。

28.4.1 自动化框架的基本特性

作为自动化专家，应该知道任何面向标准业务的应用都不可能100%自动化，因为存在各种相关因素的制约，如涉及的接口和接口技术、使用的第三方工具及其与测试工具的兼容性、实时应用的复杂性以及各种其他因素。成功自动化的一个更重要的方面就是，将所有相关的功能测试用例组合到一起，并优化可复用组件。实际上，不可能为每个测试用例（这些测试用例是为测试自动化确定的）都创建一个测试脚本。理想状态下，考虑到可复用性，自动化脚本的数量应该少于为自动化确定的测试用例的数量。

下面是测试自动化框架的一些最佳实践。

1. 定义文件夹结构

自动化项目成功的基础在于一致性和可复用性。应该采用一种组织中的每个人都能理解和访问该结构的方式为自动化项目定义文件夹结构。下面介绍一种示例格式，可根据项目的复杂性对该格式进行自定义修改。

- 项目自动化
 - 资料库
 - 驱动程序脚本
 - 可复用的窗口函数
 - 可复用的业务函数
 - 错误处理函数
 - 测试脚本
 - 常用（按模块）
 - ◆ 测试脚本
 - 登录（示例）
 - ◆ 测试脚本

- ## ■ 测试数据文件

■ 记录

• 测试报告

• 错误日志

基于测试自动化框架方法的另一个优

- 文件处理:

□ 字符串处理:

- 缓冲区处理;

- 变量处理:

□ 数据库访问:

□ 日志记录实用程序:

□ 系统/环境处理:

□ 应用映射函数:

□ 系统消息处理或系统API增强与封装。

4. 开发脚本的指导方针和评审检查表

为编写自动化脚本所定义的指导方针和标准对实施一致性、可复用性和易维护性来说是必不

的。部分应该记录的标准如下:

□ 变量;

□ 连接到数据库:

- 调用可复用的函数。

个函数最后都会检查屏幕上是否会出现错误消息。

如果发生这样的错误，脚本会记录相关信息，并继续执行下一个测试用例/测试场景。

6. 定义维护过程

为了把将来的增强合并到现有应用中，测试自动化框架应该定义相应的方法和手段。对于将来的增强来说，这个框架应该是可扩展的。框架还应该定义如何标识修改现有测试自动化套件中的功能所造成的影响，以及如何修改这些功能。

28.4.2 标准自动化框架

根据自动化测试组织的成熟度，测试自动化框架已经经过了一段时间的演进。

数据驱动框架、模块化框架、关键字驱动框架和混合框架是一些流行的框架模型，所有测试自动化领域都会用到这些框架。

337

1. 数据驱动框架

数据驱动测试是从数据文件（如CVS文件、Excel文件、文本文件等）中读取测试输入和输出值来驱动测试的框架。在不同的应用界面之间切换、读取数据文件并用日志记录测试状态和信息的这些行为都被编码在测试脚本中。

对使用不同测试数据组合（参见表28-1）的应用界面施行测试时，数据驱动测试框架是很有用的。此时，只有一种脚本可以处理各种测试组合，具体取决于数据文件中指定的测试数据的不同组合。数据文件的每一行是一个测试用例。

表28-1 测试数据

账 号	信用卡卡号	有 效 期	自动扣款	备 注
313 254 2288	2222 3333 4444	08/04/21	Y	
567 298 9988	9923 8769 8742	08/03/12	N	
987 765 9843	8769 6754 4397	09/02/23	Y	
769 457 5544	6549 7692 4214	09/01/23	Y	

28

当测试人员需要确认业务函数，而这项业务函数又具有大量相关数据的不同组合（参见表28-1）时，数据驱动框架非常有用。当人为错误在所难免时，使用这种方法非常高效，并且可以省去单调乏味的手工测试工作。使用这种方法也能大幅缩短测试周期，提高生产力。

对于这一框架需要考虑的关键就是调整测试数据，以确保测试数据的覆盖率达到最大化，从而挖掘出系统中隐藏的缺陷。（第34章将介绍Smartware Technologies公司的SmartTest™工具，该工具会自动生成测试数据。）

2. 模块化框架

模块化框架方法（如图28-4所示）需要创建小的、独立的自动化脚本和函数。这些脚本和函数代表正在测试的应用的模块、部分和函数。这些小脚本稍后被用在层次化方法中，以构建更大的测试，实现一个特殊的测试用例。

下面这些模块化格式将解释如何使用应用程序中可用的不同级别和特性来构建这种框架。

338

驱动程序脚本、主测试脚本、业务函数脚本、确认脚本、子例程脚本和报告脚本是模块化（例如零售银行业务函数）的组成部分。

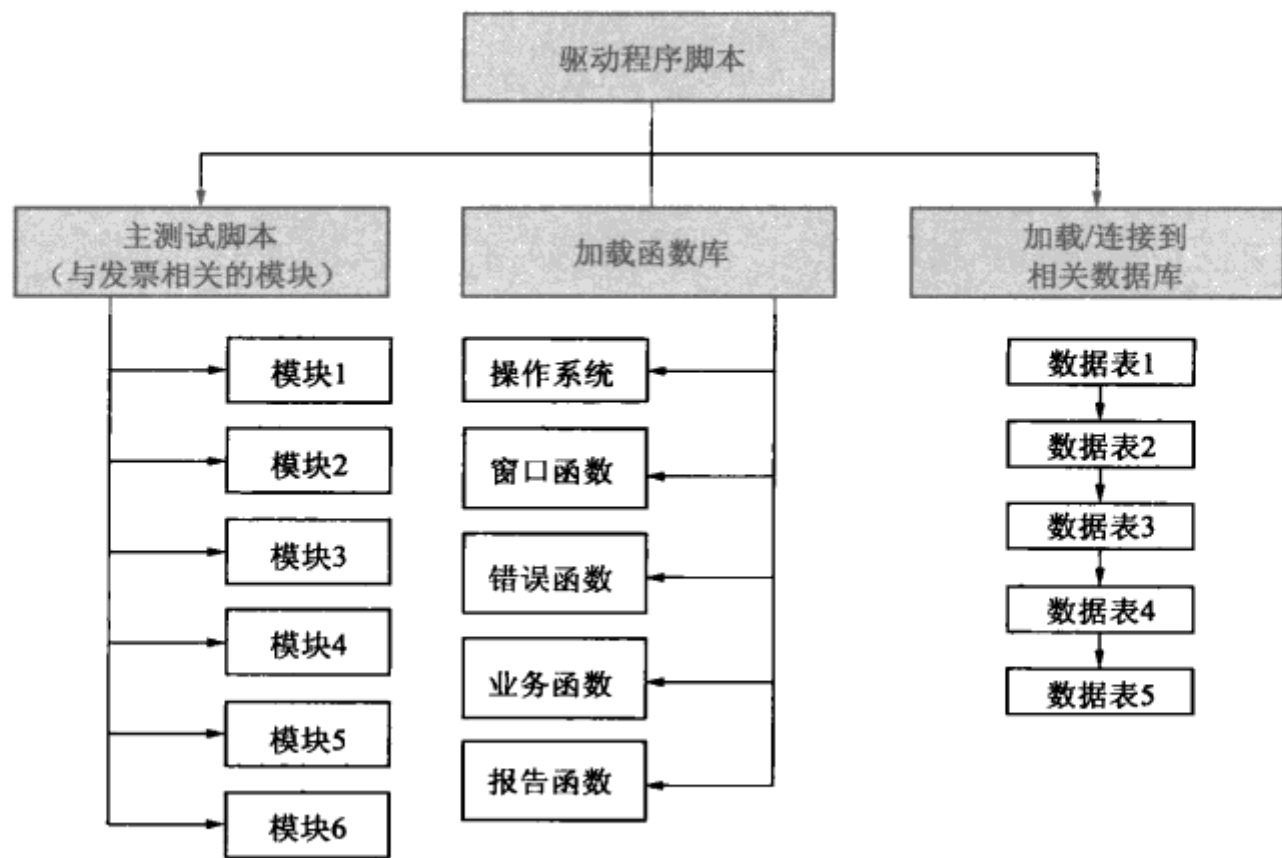


图28-4 模块化框架

下面是模块化框架的优点。

- 因为编写脚本的目的是执行和测试每个业务函数，所以它们很容易就能组合到“高级”测试脚本中，以适应复杂的测试场景。
- 减少创建自动化测试脚本的冗余和工作量。
- 开发应用程序的同时就可以开发脚本。
- 维护脚本的预期结果非常容易。
- 错误处理机制更加健壮，支持测试脚本在无人看管的状态下执行。
- 因为这些脚本相互之间的依赖性很小，所以能以即插即用方式使用。

3. 关键字驱动框架

339

关键字驱动框架是业务自动化的一种流行的模型。使用这种框架，不同界面、不同函数和不同业务组件可被指定为数据表中的关键字。这些要执行的测试数据和测试操作使用测试自动化工具编写成脚本。测试完全由数据表中指定的不同关键字驱动。由于这些关键字被映射到一个表的相关自动化脚本中，所以该框架也被称为表驱动框架。表28-2给出了一种示例格式。

表28-2 关键字测试数据

窗 口	控 件	操 作	参 数
窗口名	菜单	单击	打开
窗口名	按钮	单击	文件夹名称
窗口名		确认	结果
窗口名	菜单	单击	关闭

测试套件由所有的测试用例文件组成（如图28-5所示），用户能够为列出的所有测试用例选择具体的测试套件，以根据测试套件文件的打开或关闭标志执行测试。

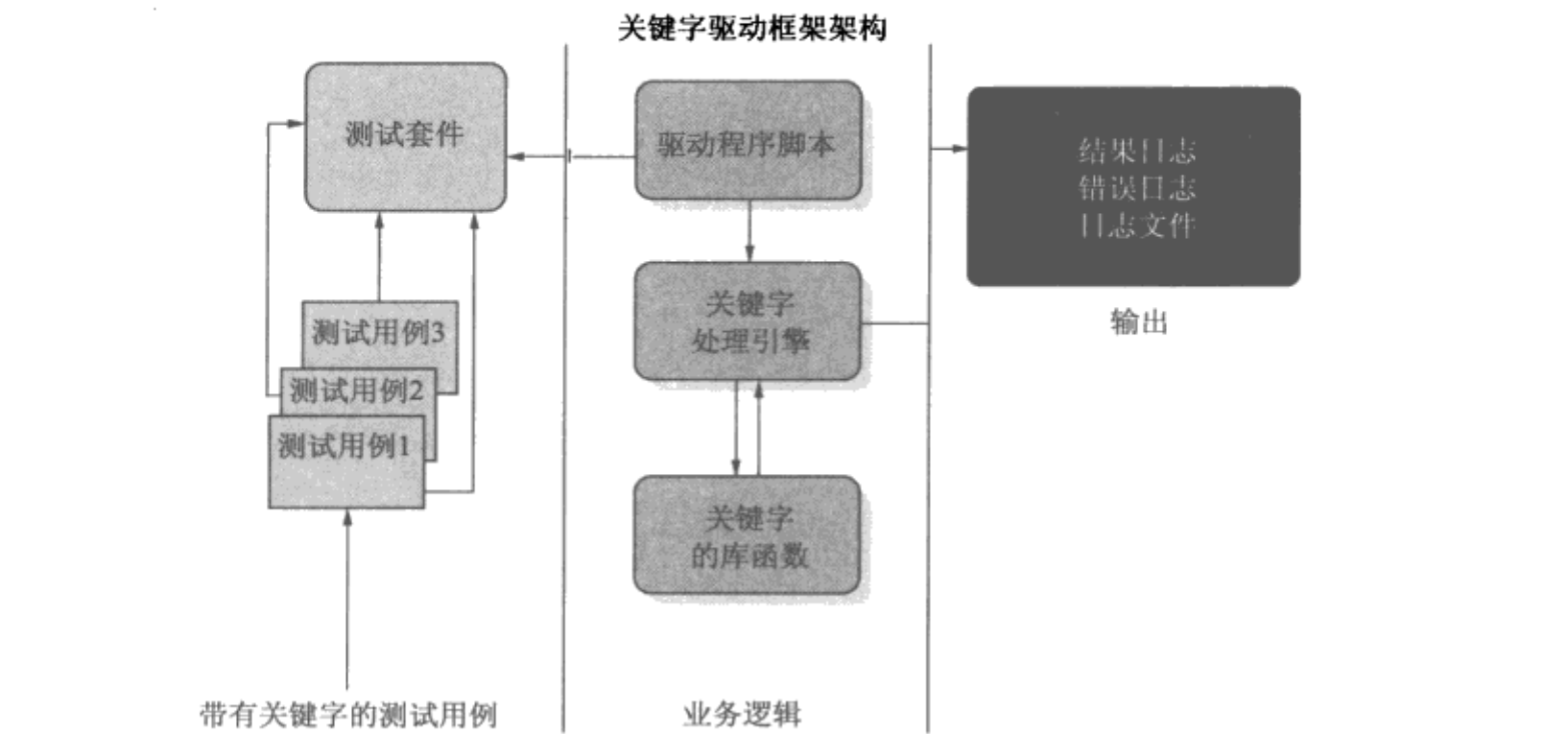


图28-5 关键字驱动框架

测试套件可以是包含多个列的Excel表，这些列分别对应测试用例ID、描述、是否执行（Y/N）、对象存储库路径、测试用例文件路径等。

测试用例文件包含执行测试用例的详细步骤。测试用例文件是Excel表形式的，其中包含关键字、对象名、参数等列。

驱动程序脚本根据操作字段中包含的关键字，从测试套件中读取测试用例文件，检查测试用例执行时每一步的关键字，并按照顺序执行每一步。这些关键字由处理引擎处理，处理引擎又会根据关键字调用适当的库函数。关键字操作在库函数中实现。执行关键字之前，驱动程序脚本会执行错误检查，并记录任何相关信息。

测试人员也可以借助于启动脚本扩展这种框架。
启动脚本完成测试设置的初始化并读取测试套件，然后调用驱动程序脚本，执行在测试套件文件中标记为执行的所有测试用例。

关键字驱动测试的优点在于，测试人员不需要理解代码的含义，就可以执行脚本。测试人员需要熟悉各种关键字和相关函数，这些关键字和函数需要确认才能执行这些脚本。自动化专家将通过代码为所需关键字创建函数。

关键字驱动框架要求每个人都具备良好的脚本编程技巧（根据测试工具），以创建关键字函数。

4. 混合框架

最常实现的框架（参见图28-6）是前面提到的所有技术的组合，并扬长避短。大多数随着时间的推移演变为多个项目的框架都是这种框架。它是由核心数据引擎、通用组件函数和函数库定

义的。而函数库提供了通用例程，虽然这些例程严重依赖于这三个元素，但是它们即使在关键字驱动框架、核心引擎和组件函数的上下文环境之外也很有用。

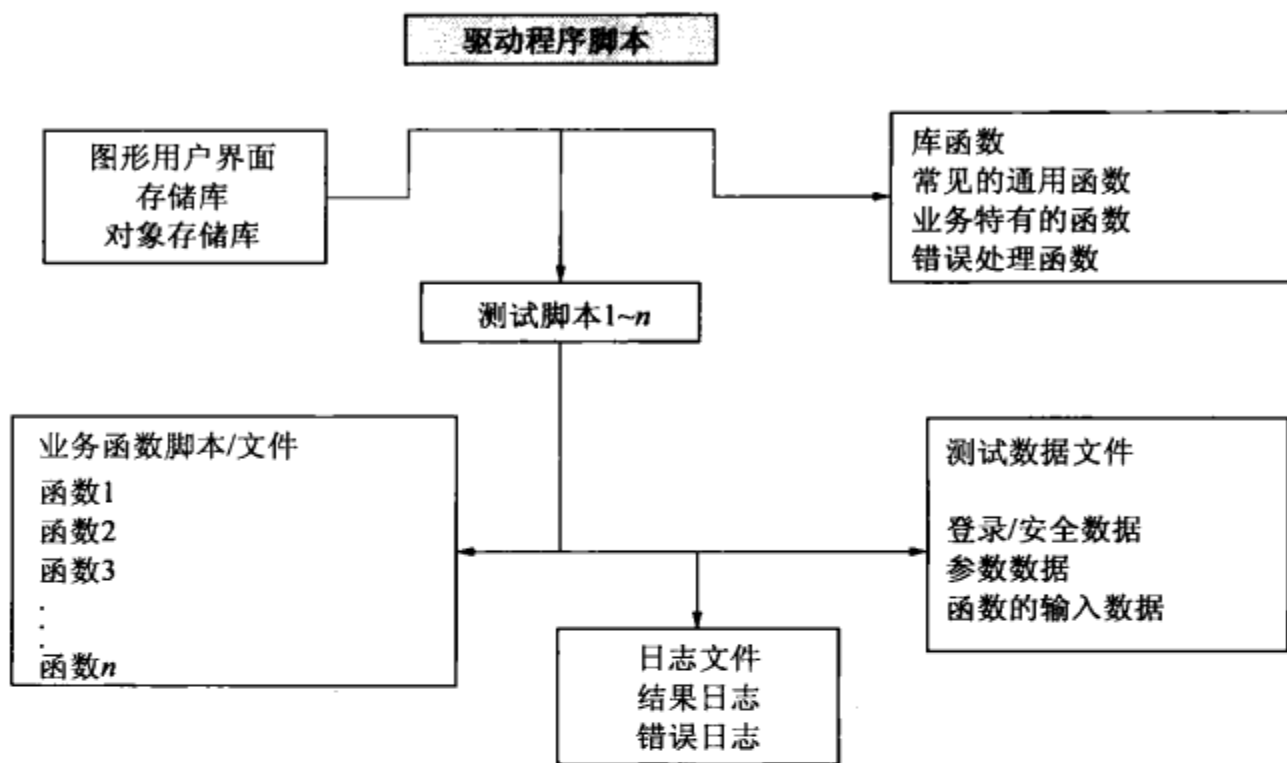


图28-6 混合测试框架

测试执行首先执行驱动程序脚本。这个脚本涉及一个或多个驱动程序脚本提供的核心数据引擎，而驱动程序脚本处理涉及每一级主要脚本的测试表。

核心数据引擎可以根据测试需求，由以下内容来实现。

- (1) 发布驱动程序。
- (2) 测试套件驱动程序。
- (3) 测试脚本驱动程序。

发布驱动程序由多个测试套件组成，测试套件由多个测试脚本组成，测试脚本由多个测试数据的集合组成。

驱动程序脚本首先调用发布驱动程序，发布驱动程序又调用相应的套件驱动程序。套件驱动程序调用各自的测试脚本。被调用的测试脚本通过为每个脚本获取相应的测试数据来执行。

构建混合的测试自动化框架要求架构师了解应用技术、接口和与应用程序交互的第三方组件以及应用程序的业务流程。应彻底地分析测试用例，以了解和标识可复用的业务组件。了解应用程序可以帮助架构师确定需要自动化应用程序的框架类型。

自动化分析人员应该确定适用于正在进行自动化的应用程序的相关框架，并设计这一框架。基础设计应该是灵活的、可扩展的。框架应考虑将来的增强和发布，并应被有效地设计，以增强模块化和可复用性。

在项目测试中,倾向于主要对系统或组件必须实现的功能进行功能测试,而忽略非功能测试。必须实现的那些功能定义了判断系统运行情况的标准。

非功能测试验证的是系统必须怎样运行,它约束的是系统的行为。非功能需求指定了功能需求中未包括的所有其他测试形式。

本章将讨论性能测试、安全性测试、可用性测试和合规性测试。

29.1 性能测试

现如今的复杂业务环境使得在各种不同架构中开发和维护的多个应用的集成成为必然。企业级应用的集成举足轻重。从业务的角度看,企业级应用的可扩展性、可靠性和性能等问题,已经提高了对性能测试和管理方面的需求。本章将介绍目前实行的各种性能测试。

应用的性能要从不同角度测量,以提高应用的可扩展性和性能。负载测试、压力测试和容量测试通常是在应用的开发阶段完成的,从而确保应用按生产预期运行。即使在实际生产过程中,也可以通过性能监视工具对应用的性能进行实时监视,以此了解性能的当前水平和影响性能的因素,从而使问题得到解决。

29.2 负载测试

负载测试就是通过模拟多个用户同时对应用程序的期望行为建模的一种实践。这种条件下的系统响应要综合多个因素来观察,如内存利用率、硬件容量利用率、吞吐量等。系统中引起任何一种不合理行为的根源,都应被发现并修复,这样才能让应用在投入生产后更好地工作。有许多基于厂商的免费软件工具都可以模拟系统中的数千个用户来协助负载测试。(详细信息参见第35章。)

29.3 压力测试

压力测试观察的是系统负载超过正常预期的情况下应用所做的响应。这种条件下,要么加大用户模式上的负载,要么让系统长时间持续执行(几小时或几天),借以测试硬件系统在压力作用下的健壮性。

29.4 容量测试

容量测试是性能测试的一种,它在数据容量提高到不正常水平的条件下观察系统的响应,用

来验证系统容量的物理限制和逻辑限制。

29.5 性能监视

在部署一个应用时，系统所有者要负责持续监视应用，关注是否出现性能降低的情况，因为这会影响业务。许多公司的Web业务都曾因为系统在线版不可用而惨遭失败，损失数百万美元，这种教训数不胜数。对影响性能的各种因素的持续监视，是通过性能监视来完成的。当出现性能降低或减慢的症状时，要及时开始采取适当的补救措施，确保系统不会突然停止运行，继而影响业务。

29.6 性能测试的方法

性能测试在整个应用测试生命周期中占据很长的一段。它需要利用有关应用技术、工具、语言、系统配置和容量信息等方面的专门技能。

344

性能测试架构师通过分析应用架构，确定应用所需要的性能测试类型。这需要与业务用户针对性能期望磋商后完成。

下面是在这期间需要实现的几项主要活动。

- 关键事务和非关键事务的区分。
- 应用响应时间的期望值。
- 事务的吞吐量。
- 高峰时的性能。

通过测试多用户应用中预期的正常时间负载和高峰时负载，可以在应用投入生产之前检测实时发生的问题。

29.7 知识获取过程

在知识获取阶段，性能团队将了解应用功能性、用户特征、系统架构以及应用设计。这个团队将与业务用户、应用开发人员、系统维护团队等各个利益相关人打交道，了解业务需求、通过开发人员来看计划中的系统的能力，以及系统运行时所期望的用户数量。这个团队还将了解需要就硬件、软件和网络连接对应用进行部署的生产环境。在某些情况下，这些事情是根据性能测试活动的结果来确定的。

下面是性能分析中计划执行的步骤。

(1) **定义测试范围。**这一步骤涉及多方面的知识，如多用户组、并发用户的数量、对不同功能进行访问的频率、对不同屏幕的各次访问之间的模拟随机思考时间、事务持续时间等。团队将判断数据库是否需要在两次测试间进行更新，更新到什么程度。测试间的数据库更新可能比较耗时，特别是对于大数据库。数据库更新经常比实际测试耗费更多的时间。团队还将定义描述系统性能的参数，如事务响应时间和事务吞吐量（页面、事务以及需要为识别瓶颈而监视的参数）。

(2) **针对性能测试创建计划。**性能测试团队应该研究测试环境，确保它模拟实时的生产环境。他们必须确认需要与业务用户磋商后进行测试的事务和应用方案，还应该确认将会降低应用性能

的公共窗口和与操作系统相关的事务。

345

根据收集的输入，性能测试团队将规划出各种输入参数的组合，以执行多个测试方案。这在测试执行期间可以根据系统响应来添加或修改。此外，性能测试团队还应针对数据库加载模式制订计划。

图29-1描绘的是一个常见的性能测试环境。

29.8 测试开发

下面是测试开发中计划执行的步骤，包括测试脚本开发、测试执行和测试分析。

(1) 开发测试脚本。测试脚本的开发需要完成下面这些活动。

- 配置性能测试工具，如测试环境中惠普公司的Performance Test Centre 8.1。
- 利用LoadRunner的Vugen（Virtual User generator，虚拟用户生成器）来记录脚本。
- 记录好脚本之后需要修改脚本来模拟复杂环境。

下面是一些示例。

- 做循环工作，使所捕获的单个活动执行起来就像许多个活动。
- 对变量进行参数化，从外部来源提供数据。数据来源包括文本文件和在测试时捕获的从应用返回的数据。
- 通过工具为数据输入准备好数据文件。
- 所有用户都将同时尝试访问一个特定的事务。

(2) 测试执行。测试执行需要完成下列任务。

- 测试数据设置。
- 对于大数据库，这是件耗时的事情。不过如果脚本已经为此提早开发好了，就可以直接运行脚本并加载数据。
- 利用测试工具设置测试方案。
- 启动服务器监视器，监视CPU、内存等。
- 利用LoadRunner的Controller生成虚拟用户，通过用户负载重播脚本。这个控制器将记录测试结果，测试结果输出后用于进一步分析。
- 在不同的用户负载下执行测试脚本。然后在几次执行之间通过运行数据库加载脚本来更新数据库。

(3) 分析。LoadRunner有用于分析和报告的标准报告，下面是生成的部分报告。

346

- 事务性能总结报告（Transaction Performance Summary Report）。
- 按虚拟用户生成的事务详细报告（Detail Transaction Report By vuser）。
- 按虚拟用户生成的事务性能报告（Transaction Performance By vuser Report）。
- 方案执行报告（Scenario Execution Report）。
- 失败事务报告（Failed Transaction Report）。
- 数据库服务器报告监视器（Database Server Report Monitors）。
- 网络延迟监视器（Network Delay Monitors）。
- 系统资源监视器（System Resource Monitors）。

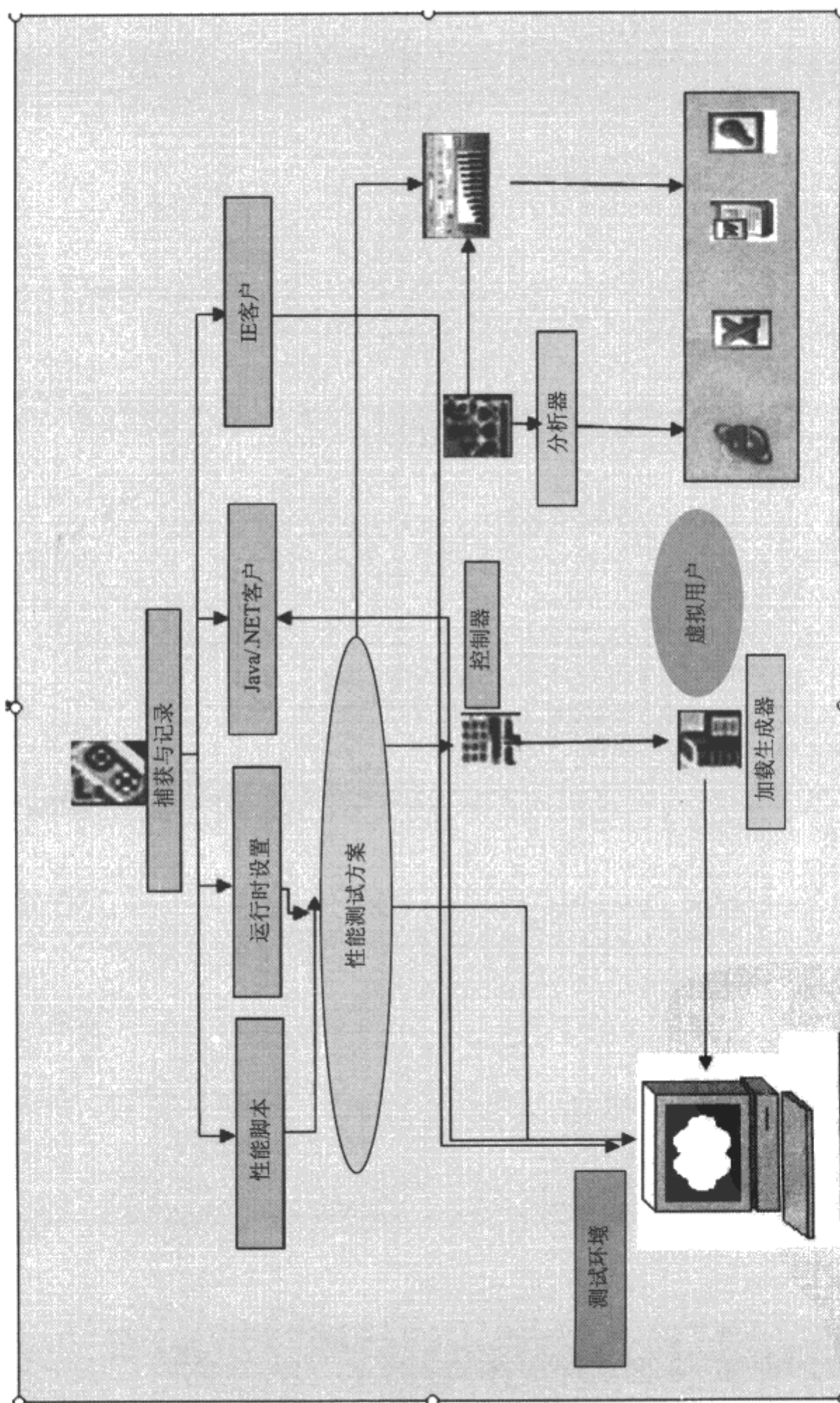


图29-1 性能测试环境

分析团队分析从生成的报告中收集的数据，从而找到瓶颈的位置。

下面是借助一个性能测试工具得到的一些常规的输出，这个工具能帮助性能测试分析师理解和分析性能需求。

- 测试总结报告。这是一个测试总结报告，如图29-2所示，这个报告展示了性能测试的整体结果，考虑的因素包括系统虚拟用户数量、以字节为单位的总吞吐量、平均每秒的吞吐量、系统的总命中数、性能测试中给定的每个事务每秒的命中数。这个报告将为性能测试分析师提供有关性能参数的指标。

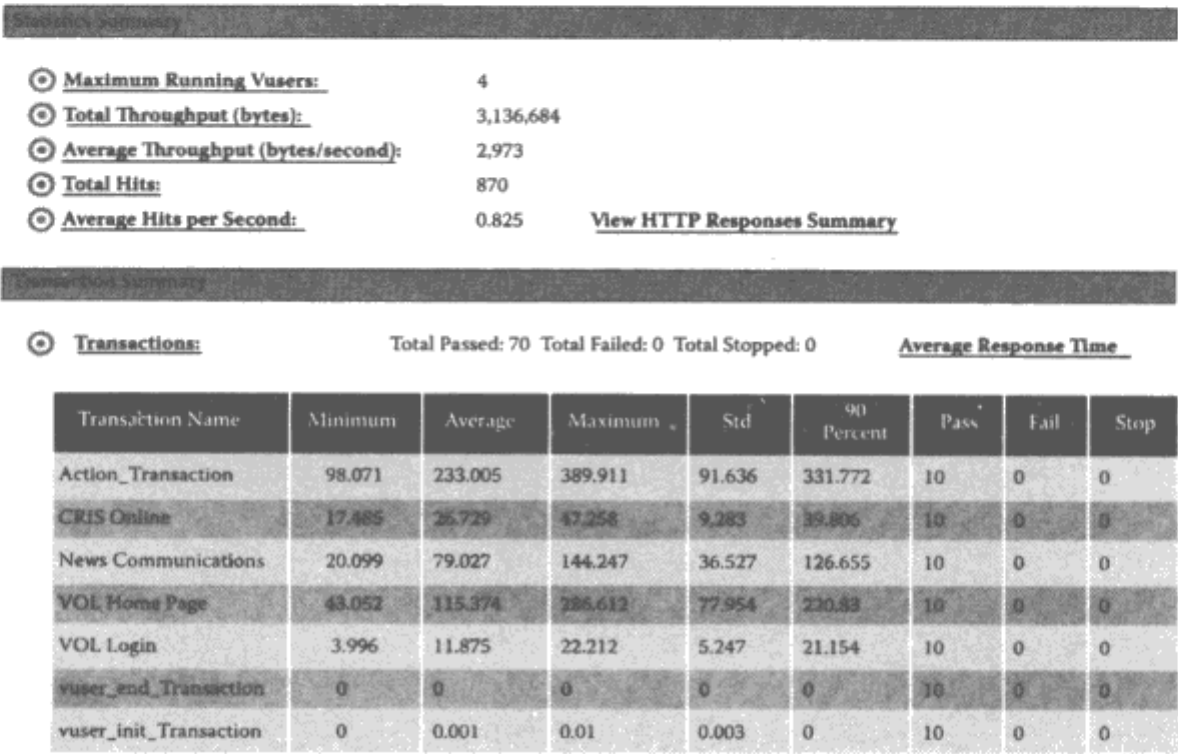


图29-2 测试总结报告

- 事务平均响应时间。图29-3显示的是给定方案在不同时间点的平均响应时间。根据系统用户数量和系统吞吐量的不同，这个时间会有所不同。

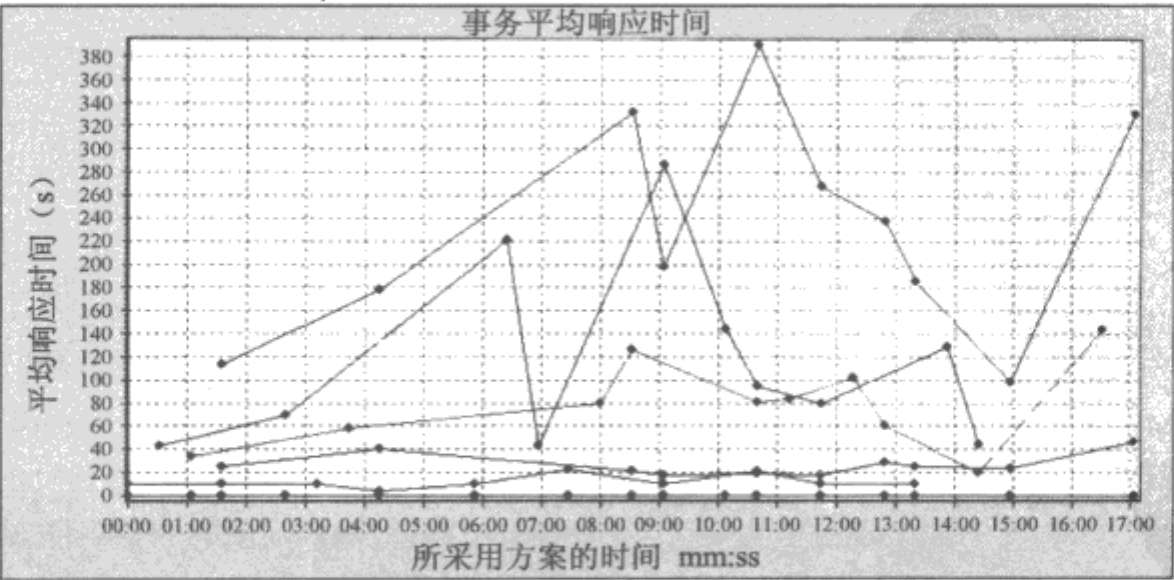


图29-3 事务平均响应时间

- 不同负载下的事务平均响应时间。图29-4展示了加大系统负载时，事务平均响应时间是如何变化的。分析师将会了解加大负载会怎样影响响应时间，运行中的系统可容忍的极限是多少。

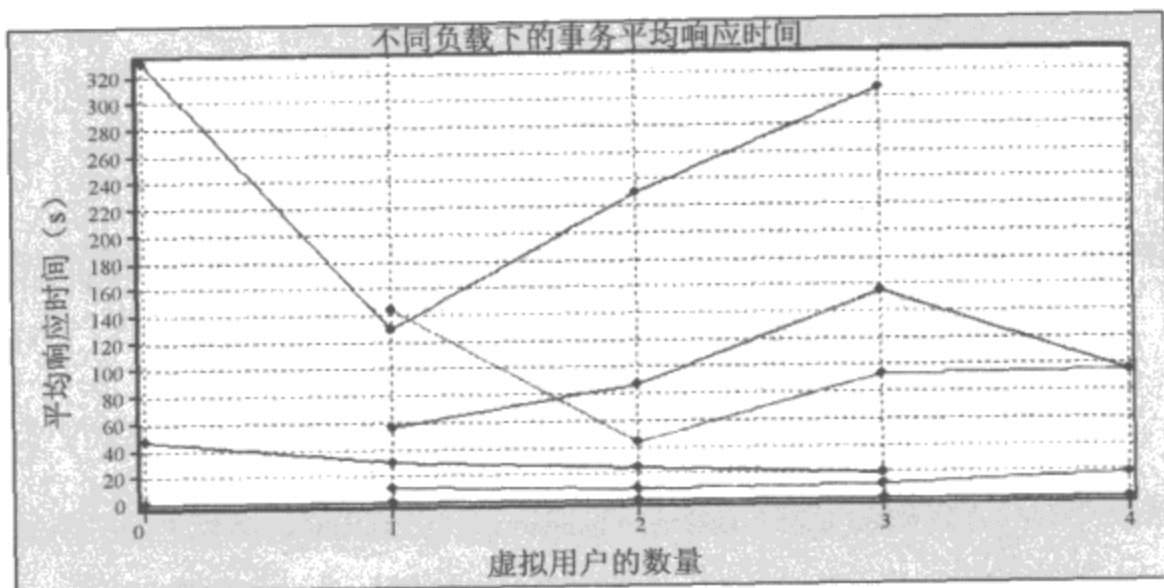


图29-4 不同负载下的事务平均响应时间

- CPU利用率。分析师从CPU利用率图可以知道，在指定事务的不同点，CPU利用率是多少。这个图将帮助分析师设置理想的CPU利用率水平。
- 页面组件百分比分析。图29-5中的饼图展示了每个页面组件在平均下载时间（以秒为单位）之和中所占的百分比。

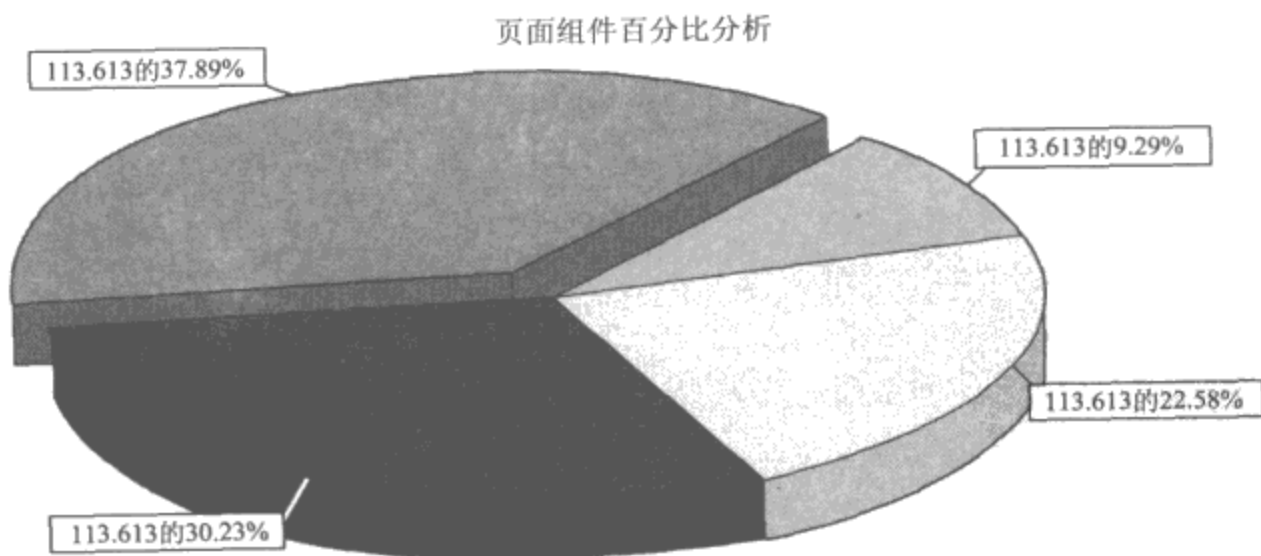


图29-5 页面组件百分比分析

- 网络延迟时间。网络延迟时间由网络传播延迟时间、序列化延迟时间、排队延迟时间组成。传播延迟时间是物理信号穿越传播路径花费的时间，序列化延迟时间是实际传输分组时花费的时间，排队延迟时间是一个分组在路由队列中花费的时间。图29-6是一个网络延迟时间图。

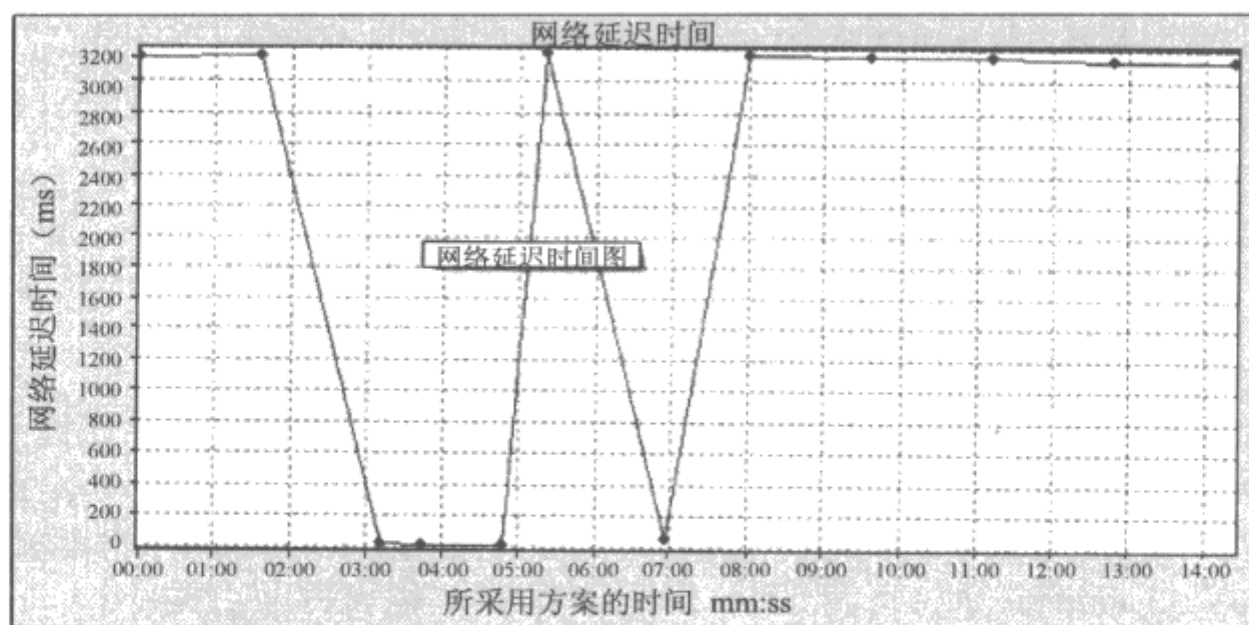


图29-6 网络延迟时间图

29.9 性能测试可交付物

性能测试团队在性能测试期间负责完成下面的可交付物。

- 性能测试策略。
- 性能测试计划。
- 选定的性能测试方案。
- 所选定方案的Vuser（虚拟用户）脚本。
- Vuser脚本文档。
- 测试执行计划。
- 测试数据信息报告。
- 测试运行报告（每日报告）。
- 分析结果。
- 性能测试报告。

团队提供的建议也将作为性能测试报告的一部分提交。例如，如果性能测试团队发现测试架构不够好，他们就可以推荐新的架构，并说明新架构所支持的用户数量。

29.10 安全性测试

安全性测试曾一度被认为是网络管理人员和系统开发人员完成的一项技术工作，应用的安全性在软件开发生命周期的测试阶段没有得到足够的重视。安全事故数量不断攀升，企业所有者对因安全问题导致的无效应用逐渐提高了警觉，于是，安全性测试开始移到了软件测试人员的职责范围中。Gartner的报告表明，四分之三的Web网站都很容易受到攻击，75%的黑客现象发生在应用级别。全球越来越多的客户开始将应用的安全性纳为软件测试的一部分。

安全性的基石是保密性、完整性和可用性。对于关键的应用，需要为不同的用户提供不同的访问级别。事务的安全性保证了客户的保密性，这是成功实现应用的关键因素。按照SOX的第404

部分，组织要对金融报告进行内部控制，这就需要测试应用的完整性。

下面介绍一下成功安全性启动的步骤。

29.10.1 步骤 1：确定安全性测试的范围

安全性测试有下列两个主要目标。

- 验证应用是否满足安全需求。
- 识别应用在给定环境下的安全漏洞。

对Web应用进行全面的安全评估是一项复杂的任务，与其他软件分析任务类似，需要应用一种方法论、完整的测试规程和一套有用的工具，还需要一定的技巧和知识。手动渗透测试和自动工具都可以用于查找Web应用中的重要安全漏洞。开发所用的技术和应用程序的漏洞决定着自动扫描和手动渗透测试两者之间的正确平衡，从而提供最佳的Web应用安全覆盖率。

安全性测试首先进行的是漏洞评估。漏洞扫描能检查出启用了IP设备的网段、枚举系统、操作系统和应用程序中是否存在安全漏洞。除了识别操作系统版本、IP协议和所监听的TCP/UDP端口号之外，漏洞扫描还能识别常见的安全威胁，如弱密码、访问权限宽松的文件、安全配置问题等。

应该为应用或产品的每一个阶段（如开发阶段、实现阶段、配置阶段、运行阶段和维护阶段）开发安全测试策略，而且，安全测试最好由独立的测试团队完成。测试目标要通过威胁模型来确定，界面（如用户界面）、套接字、文件输入、API、邮件配置和设备都应包括在这一范围内。像网络带宽、内存、磁盘空间、文件和套接字这样的性能瓶颈，都应该接受安全性测试。

352

29.10.2 步骤 2：生成测试用例并执行

应用程序的安全性要通过试图违反内置的安全控制来测试。这项技术能确保系统中的保护机制足够完善，保证应用程序不会受到不正当的或未经授权的访问。测试人员通过持续请求使系统过载，因而拒绝将服务交给其他人。测试人员可能故意使系统出错，以破坏恢复期间的安全性，也可能浏览不安全的数据，从而找到进入系统的关键，下面是需要进行安全性测试的部分：

- 用户身份验证；
- 密码管理；
- 接入验证；
- 异常处理；
- 安全数据存储和传输；
- 日志记录；
- 监视和警告；
- 修改管理；
- 应用开发；
- 阶段性安全评估和审计。

针对缓冲区溢出、SQL注入、跨站点脚本、参数篡改、cookie病毒入侵、隐藏字段、调试选项、无效输入、被破坏的授权信息和身份验证信息、会话管理等进行安全性测试时，应该生成测

试用例。理想情况下，安全性测试应该在功能集成测试和性能测试的最后阶段执行。这样有利于检测应用中隐藏的安全威胁。

完成安全性测试之后，应在报告中总结测试结果。总结报告应包含一些细节信息，如采取的测试类型、识别出的安全风险及其级别，这些都有助于企业对应用部署作出决定。

29.11 安全性测试的类型

下面列举了安全性测试的类型及各自的目的、工具和方法。

29.11.1 网络扫描

网络扫描涉及通过端口扫描判断是否所有主机都有可能与组织网络连接。这能识别出所有活动的主机和打开的端口，有些扫描程序还能提供关于被扫描主机和在特定端口运行的应用的额外信息。网络扫描应该在系统中持续执行。

1. 目的

- 检查连接到的未授权主机。
- 标识易受攻击的服务。
- 根据安全策略判断与所允许的服务的偏差。
- 在渗透测试中提供帮助。
- 协助指令检测系统的配置工作。

2. 工具

- Fscan——扫描TCP和UDP端口的一个命令行端口扫描程序。
- LANguard 网络扫描程序——扫描免费软件安全性和端口的扫描程序。
- DUMPsec——用于微软公司的Windows的安全审计程序。

3. 方法

必须具备相当水平的专业知识的人才能解释测试结果。扫描操作可能因为消耗过多的带宽和过少的响应时间而导致网络运行中断。测试结果要用文档记录下来并作出分析，接下来采用纠正步骤。下面是一些可选用的方法。

- 调查并断开未经授权的主机。
- 禁用或删除不必要的和易受攻击的服务。
- 修改防火墙设置，限制外部访问。
- 修改易受攻击的主机来限制对易受攻击服务的访问。

网络扫描的速度和效率取决于系统中主机的数量，还有许多自动的免费软件工具可以使用。网络扫描工具的缺点在于它们不能直接识别漏洞。

29.11.2 漏洞扫描

除了扫描端口之外，这些测试工具还能报告相关的漏洞，识别过时的软件版本、未加利用的补丁和系统更新、与整个组织的安全策略有偏差的地方等。漏洞扫描的副作用是，那些工具常常加载系统和持续更新漏洞数据库以便捕获它们。

1. 目的

- 识别活动的主机（连接到因特网的计算机）。
- 识别主机上活动的易受攻击的服务，如邮件服务。
- 识别应用、错误配置和操作系统。
- 验证与主机应用的安全策略的合规性。

2. 工具

- Cybercop扫描程序——一种基于网络的漏洞测试工具。
- ISS因特网扫描程序——一种识别安全问题的漏洞扫描工具。
- SecureScan——NX、SAINT、SARA是另外一些漏洞扫描工具。

3. 方法

漏洞扫描需要确认操作系统和主应用程序的安全补丁和软件版本是实时更新的。测试结果需要记录下来并分析。

下面是推荐采用的纠正方法。

- 升级易受攻击的系统或为其打上补丁。
- 加强配置管理。
- 利用专门的资源来监视漏洞。
- 对整个组织的安全策略和架构保持持续改进。

网络扫描的速度是可以提高的，这取决于所扫描的主机的数量，另外还可以采用自动免费软件工具来加速。这些扫描程序进行定期扫描很容易。有时可能出现误报，这种情况需要通过结果分析才能判断出来。

29.11.3 密码破译

密码破译是指通过解释网络中的密码散列来验证用户是否采用了强密码的过程。

密码破译程序需要在系统上按月执行，甚至持续不断地执行，从而确保整个组织的正确密码组合。

工具

- Crack 5——UNIX密码破译程序。
- John the Ripper——Windows和UNIX密码破译程序。
- L0phtCrack——Windows密码破译程序。

如果被破译的密码是按照策略选定的，那么应该修改原来的策略，减少可被破译密码的比例。如果被破译的密码不是按照策略选定的，那么应该指导用户按照策略设置密码。

29.11.4 日志评审

各种系统日志被用来判断与组织的安全策略的偏离之处，如防火墙日志、IDS日志、服务器日志，以及用来收集系统和网络上的审计数据的任何其他日志。审计日志可以用来确保系统运行遵循策略。

手工审计日志的评审工作很烦琐，而且费时。自动审计工具提供了一种明显缩短评审时间的

途径，而且会生成预定义的和定制的报告，针对一系列特定活动对日志内容进行总结。

方法

举个例子，如果一个IDS传感器受防火墙（在程序集中）保护，那么它的日志可以用来检查服务请求和被防火墙允许进入网络的通信。如果传感器注册了防火墙不允许的未授权活动，那么将会有提示，表明防火墙的配置不再安全，网络上存在一个后门。

29.11.5 文件完整性检查器

文件完整性检查器是一种识别工具，用于识别对文件的修改，特别是未经授权的修改操作。文件完整性检查器计算并存储每个受保护文件的校验和，另外还为文件校验和建立一个数据库。所存储的校验和应该能定期重新计算，以测试存储值所对应的当前值，便于识别任何文件修改操作。参照数据库应该离线存储，才能不让攻击破坏操作系统和通过修改数据库来隐藏轨迹。

1. 目的

- 识别对文件未经授权的修改。
- 在可能出现泄漏时锁定可能的损失范围。

2. 工具

- LAN guard。
- Tripwire。

356

29.11.6 病毒检测器

如果与因特网连接，使用可移动的媒体（如软盘和CD-ROM），使用共享软件或免费软件，那么任何组织都有感染计算机病毒、遭遇木马程序和蠕虫攻击的风险。如果使用了恶意代码，也存在泄漏或丢失敏感信息和保密信息的危险。为了检测病毒，需要在网络和机器上安装防病毒软件。防病毒软件应该有一个实时更新的病毒标识数据库（也叫病毒特征，帮助识别所有病毒）。在检测病毒时，防病毒软件会将文件内容与已知的计算机病毒特征比较，识别出被感染的文件。如果可以，则将感染文件隔离然后修复，否则就删除它们。一些更高级的程序还会查找行为像病毒的活动，希望识别出无法通过当前的病毒检测数据库识别出来的新病毒或变异病毒。

1. 工具

- McAfee。
- Symantec。
- Trend Micro。

2. 方法

防病毒程序主要有两类：安装在网络设备上的和安装在终端用户机器上的。

安装在网络设备上的病毒检测器通常安装在邮件服务器上，或者与一个组织的网络边界上的防火墙协同工作。基于服务器的病毒检测程序可以在病毒进入网络之前或用户下载电子邮件之前将它们检测出来。

另一类防病毒程序安装在终端用户的机器上，这种软件检测的是存在于电子邮件、软盘、硬盘、文档中的恶意代码，不过只针对本地主机。有时候这类软件也检测Web站点上的恶意代码。

29

29.11.7 渗透测试

357

渗透测试是一种安全性测试，评估人员根据自己对系统设计和实现的了解，尽力避开系统的安全特性。有一点很重要，那就是判断组织网络易受攻击程度，以及网络发生泄漏时可能遭受的损失级别。渗透测试可以设计成专门对内部攻击行为或外部攻击行为进行模拟。如果内部测试和外部测试都需要执行，那么外部测试通常优先实施。执行外部渗透测试时，防火墙通常会限制从外部源进入内部网络的流量数目及流量种类。根据允许使用的协议，一开始的攻击行为一般会关注经常使用的应用程序协议，如FTP、HTTP、SMTP和POP。

1. 目的

渗透测试的目的是通过常见工具和攻击者所采用的技术，来确定获取系统访问权的方法。这类测试能暴露内核代码、缓冲区溢出、符号链接、文件描述符、竞争状态、文件和目录权限中的漏洞以及木马程序等。

2. 方法

渗透测试可以公开进行，也可以隐蔽进行。这两种渗透测试通常分别被称为Blue Teaming和Red Teaming。Blue Teaming执行的渗透测试需要了解组织IT人员并征得其同意，Red Teaming执行的渗透测试不需要了解企业IT人员，但需要了解上层管理人员并得到上层管理的许可。如果不仅需要测试网络安全性，还需要测试IT人员对已观察到的安全事故的反应，并测试他们的专业知识及整个组织的安全策略的实现情况，那么这种方法适得其所。在Red Teaming中，渗透测试可以有警告，也可以没有警告。

为准确模拟真实的外部攻击，测试人员事先对目标环境一无所知，只知道目标IP地址或地址范围，并且必须在发起攻击之前秘密收集信息。他们从公共的Web网页、新闻站点和其他类似站点上收集目标信息，然后使用端口扫描程序和漏洞扫描程序识别目标主机。因为他们最可能会通过防火墙检查，所以进行内部攻击测试时信息量远比他们可以获取的少。在识别出可以从外部连接的网络上的主机之后，他们会试图泄漏其中一个。如果泄漏成功，他们就会利用这种访问能力泄漏其他通常不能从外部访问的主机。[参考美国国家标准技术研究所(NIST)编写的*Guideline on Security Testing*，特殊文献800-42。]

29.12 易用性测试

358

商业Web站点用户数量的不断攀升，影响着应用和用户的使用模式。当正在登录的用户数量超过预期时，系统应用的性能会受到影响。我们已经看到，通过对性能测试结果进行解释并依其行事，可以让性能提高。类似地，因为Web应用的蓬勃发展而突然出现的另一个问题是易用性。易用性测试将帮助我们评估系统终端用户访问应用时的难易程度。

根据ISO 9214-11，易用性是指在满足特定的有效性、高效性和用户满意度的条件下，一个产品能被任何特殊用户用于达到特殊目的的程度。易用性是影响一个产品或系统的用户体验的所有因素的综合。Web站点可用性测试有很多种方式，其中一个简单方法是想象一个真实用户坐在一台PC前，完成一个Web站点上的几项工作，还要记录搜索结果。如果有大量不同的用户都经历这一过程，网站的弱点就会反映出来了。

下面是易用性的3个主要原则。

- 准确清晰地沟通用户才能明白你的意思。用户只愿花很少的时间来启动对Web站点的访问，所以，必须快速地让用户相信这个站点是值得访问的。
- 提供用户想要的信息。用户必须能够很容易判定你的服务是否满足其要求，以及为什么自己应该与你打交道。
- 提供简单一致的页面设计、清晰的导航，还有一个信息架构，将信息放到用户认为能找到的地方。

不要将易用性与功能性混淆。由于功能性纯粹只涉及产品的功能和特性，因此它与使用产品时的难易程度毫无关系。

29.13 易用性测试的目的

易用性测试的目的是弄清用户的需要和期望。它通过检查测试的应用来判断目标用户使用测试过的应用时能否满足自己的要求或达到自己的目的。

下面是易用性测试的一些关键要素。

(1) 系统状态的可见性。系统应该在合理的时间内，通过恰当的反馈，让用户知道正在发生的事情。

(2) 让系统与真实世界匹配。系统用的应该是用户语言，有词、短语和用户熟悉的概念，而不是面向系统的术语。遵循真实世界的惯例，按自然的逻辑顺序显示信息。

(3) 便于用户学习。对于一个以前从未用过的系统，用户需要很快就能学会用它来完成基本任务。

(4) 使用时的灵活性和高效性。采用不同的途径以高效方式使用系统是非常重要的。

(5) 加速器。这通常会提高专家用户的交互速度。无论是没有经验的用户，还是经验丰富的用户，系统都能迎合他们的需求。加速器使用户可以进行频繁的操作。

(6) 一致性。具有一致性的操作在类似的情形下会产生相同的反应。例如，单击一个超链接将会打开一个弹出窗口，而单击一个按钮会打开一个新屏幕。

(7) 出现错误的频率和严重性。系统中出现错误的频率是多少？有多严重？用户如何从错误中恢复？与其发送一条好的出错消息，不如认真设计程序，从一开始就阻止问题发生。可以消除可能出错的条件，也可以检查它们，在用户提交操作之前向他们显示确认选项。

(8) 美观简洁的设计。对话框不应该包含无关信息或很少需要的信息。一个对话框中每个额外的信息单元都会与相关信息单元竞争，减小它们的相对可见性。

(9) 图形用户界面。图形用户界面指的是前端，或者是软件应用或Web站点中用户能看见并加以操作的那部分。

(10) 方向定位。用户通过方向定位能知道自己在应用或Web站点中的位置。在用户进一步导航和了解应用时，以及更正导航错误时，方向定位很关键。

29.13.1 方法和执行

易用性专家应该确定在系统使用方面会影响和可能影响用户的事务。应该为以下领域编写测

试用例。

- 站点设计和网页设计。
- 导航辅助和普通的观感。
- 页面大小、文件大小、页面尺寸调整。
- 字体对易读性的影响。
- 使用文本元素和格式列表、块文本以及表。
- 提高Web网页的可访问性。
- 何时使用图片以及如何让图片更有效。
- 链接的外观以及使用链接的位置和方法。
- 提高用户的高效性。

360

易用性专家可以按照与功能性测试用例相似的格式来编写测试用例。要执行这些测试用例的易用性专家应该对Web应用的使用模式有基本了解，还应该用文档记录预期的结果。

一般情况下都需要挑选不同行业的用户来访问系统，执行特定操作并记下他们在易用性测试中的用户体验。这样能尽量真实地反映现实情况。易用性测试应该在实时系统、纸上原型或演示应用上进行。基于审查的用户测试的一种最有效的方式是使用“易用性检查表”。基于可用性检查表的用户测试实现起来成本极其低廉，只需要非常少的测试人员。

易用性测试人员可以是随时停下手中的事情来充当执行测试的志愿者。测试人员应该毫无顾忌地表达自己的想法，不用担心会伤害产品开发人员的感情，即使他们的错误可能意味着开发人员必须付出更多的劳动。你也许认为这种测试工作很简单，也可能对此感到厌烦，但测试人员必须严肃对待这项工作。

29.13.2 易用性测试的原则

易用性专家要清晰地记录为易用性测试用例准备的各种原则、对测试中外部用户的定义、易用性测试的执行原则。下面是一些标准原则。

- 除了最简单的和最不正式的测试之外，所有的易用性测试都要首先进行先导测试（pilot test）。
- 要让测试人员感到轻松，任何观察都应该通知他们。你至少要参与一项测试，亲身体会测试人员和其他参与者所承受的压力。
- 保证参与者和测试人员有权放弃他们无法完成的任何任务。
- 除非确实必要，否则不要增加参与者。
- 尽可能详细地记录事件，如果可能，详细至键击和鼠标单击。
- 如果有观察者，无论如何不能让他们对测试产生干扰。
- 开发人员可能会因为他们观察到的事情和你的报告而感到不安，你应该对此保持关注。

29.13.3 可访问性测试和《康复法案》第 508 条

1998年，美国国会修订了《康复法案》（Rehabilitation Act），要求联邦机构所采用的电子信息技术不能对残障人士形成障碍。无法访问的技术会妨碍个人方便快捷地获取和使用信息的能

力。已颁布的《康复法案》第508条消除了信息技术的壁垒，为残障人士争取了新的机会，并且鼓励有助于实现这一目标的技术的开发。这项法案在所有联邦机构开发、实现、维护或使用电子信息技术时都适用。法案的第508条（29 U.S.C. '794d）规定，联邦必须让残障雇员和大众一样能够获取其他人所能获取的信息。Web可访问性意味着残障人士也应该能够使用Web，更准确地说，意味着他们应该能够感知、理解和浏览Web，并与之交互，还能为Web作出贡献。Web可访问性也能使其他人受益，包括因为年龄增长而能力衰退的老年人。

这些标准定义了所涉及技术的种类，在第4项条款设立了可访问性的最低级别。应用部分（1194.2）概述了标准的范围和覆盖面。这些标准覆盖了联邦部门所有的电子信息技术，包括用于通信、复制、计算、存储、显示、控制、传输和生产等方面的技术，涉及计算机、软件、网络、外部设备和其他类型的电子办公器材。这些标准还对电子信息技术作出了定义，规定在某种程度上，电子信息技术是指“用于创建、转换、复制数据或信息的任何设备、互联系统或设备子系统”。

这些标准还规定了针对某些技术的准则，包括以下几项。

- 软件应用和操作系统。
- 基于Web的信息或应用。
- 电信产品。
- 视频和多媒体产品。
- 独立的封闭式产品（如信息咨询台、计算器和传真机）。
- 台式计算机和便携式计算机。

这一条声明了所涉技术功能化能力的技术规约和性能需求。从这种双重方法认识到了所涉技术不断发展、动态变化的本质，并认识到需要有清晰、具体的标准来促进合规性。有几项条款是专门用于保障自适应设备兼容性的，残障人士一般利用这些设备访问信息和进行交流，如利用屏幕阅读器、盲文显示器和电传打字机等。

大部分软件规约都与服务于视障人群的产品可用性有关。例如，有一项条款要求产品必须有可选的键盘导航方式，对于无法依赖定点设备（如鼠标）的视障人群来说，这是必不可少的。还有其他一些条款规定了有关动画显示、颜色对比度设置、闪烁频率、电子表格以及其他方面的要求。

基于Web的技术和信息方面的标准是以万维网联盟的Web Accessibility Initiative提出的Web访问指导原则为基准的。这许多的条款确保了视障人群的访问能力，他们需要依赖辅助产品来访问计算机技术，例如，屏幕阅读器就可以将计算机屏幕上的信息自动转换为语音输出和可刷新的盲文显示，便于他们使用。有些规定是必需的，例如图形和格式化装置（如帧）的文字标签或标识，它们让这些装置能够以用户可感知的方式“读”其信息。这些标准没有禁止使用Web站点图形和动画，法案的目的只是为了确保这些信息也能方便获取。通常这意味着需要为图形或某些格式元素使用文本标签或描述符。（HTML代码已经为图形提供了一种“Alt Text”标签，可以用作图形的文字描述符。）《康复法案》的第508条还声明了有关多媒体展示、图像地图、样式表、脚本语言、applet和插件及电子表格的易用性。

这些标准适用于所有联邦Web站点，但私营部门除外（除非站点与联邦机构签署过合同，只有合同约定范围内的Web站点或部门必须遵守这些标准规定）。易访问的站点带来的显著优点远

远不只是“能够访问”。例如，有“只限于文本”这一选项的站点就提供了更快的下载途径，并且便于将Web数据传输到手机和个人数字设备。

这一条标准的主要目标是，确保有听觉障碍的人有访问能力。例如，标准规定了与助听器、人工耳蜗、辅助听觉装置和电传打字机等兼容性。电传打字机是指能帮助有听觉障碍或言语障碍的人通过电话进行交流的设备，通常包括电话听筒的声音耦合器、简易键盘、可视的信息显示设备等。对于既提供语音通信功能也提供电传打字机功能的电子通信产品，需要有一个标准的非声波电传打字机连接点。还有一些规约则指定了另外一些要求，如可调节的输出音量控制、具备与助听技术连接的产品接口、视障人群或行动不便的人所用的键或控件的易用性。

顾名思义，多媒体意为多种媒体的综合，包括（但不仅限于）视频程序、演讲幻灯片产品、计算机特技。法案条款对标题解码器电路（对于屏幕大于13英寸的任何系统）和电视调谐器（包括计算机中使用的调谐卡）的二级音频通道作了规定。标准还要求联邦机构开发或采办的有关培训和信息的多媒体产品具备文字说明和音频描述。标准还规定，观察者要能够开启和关闭文字说明或视频描述特性。

363

第508条的约束对象是内嵌了软件的产品，这些产品通过内嵌软件不让用户轻易附加或安装辅助技术。例如，信息台、信息事务机、复印机、打印机、计算器、传真机以及其他类似产品，都属于这一类产品。标准要求访问功能内置到系统中，这样，用户就不必附加辅助设备了。有些规约声明了私人助听设备（电话听筒或标准的耳机接口）、触摸屏、声效输出和可调节的音量控件以及可用范围内控件位置等的机制。

第508条还对键盘和其他的机械操作控件、触摸屏、生物标识方式的使用、端口和连接头作出了规定。

第508条中提到的性能需求针对的是所有的产品评估，以及在B部分的技术标准中没有特殊要求的技术或组件。这些标准专门用于确保各个可使用的组件协同工作，创建出可用的产品。标准覆盖了包括输入功能和控制功能在内的操作、机械机制的操作以及对可视和可听信息的访问。这些条款的形成是为了让存在感觉障碍和听觉障碍的人可以找到、识别和操作输入功能、控制功能、机械功能，并且访问所提供的信息，如文本、静态或动态图像、图标、标签、声音或附带的操作线索。例如，有一项条款就要求至少提供一种模式供弱视（视觉灵敏度在20/70到20/200之间）的人使用，让他们不必依赖音频输入。这是因为，许多弱视的人的听力也有损伤。

这些标准还阐明了所涉技术中为终端用户（如联邦雇员）提供的所有信息、文档和其他支持的访问问题。这包括用户指南、终端用户安装设备的安装指南以及客户支持和技术支持交流。这些信息必须能够以多种可选方式免费获取。可选的交流方式包括盲文点字法、盒带录音、大号字体、电子文本、因特网资讯、电传打字机访问、视频材料的说明文字和音频描述等。

美国政府Web站点上提供了一套标准的测试用例，可以用来指导可访问性测试。（参见<http://www.section508.gov>。）

29.14 合规性测试

合规性测试是判断特定的实现规约的产品实现是否完成了指定的所有强制性元素，这些元素是否可操作。

合规性测试可能因时间的推移变得越来越严格，特别是随着特定的实现规约逐渐完善，这一点尤为明显。无论软件审计如何开始，这一过程都很难预测，而且经常导致流失宝贵的资源。除了资源紧张以外，软件审计还需要额外的开销来部署资产管理服务，从而防止未来违反合规性。本章介绍了一个风险评估调查，帮助判断你的组织是否遵守软件合规性审计，以及它面对的是什么级别的风险。

364

下面是6个基本步骤，让软件管理能够确保具备满足审计要求的文档。

- (1) 评审现有的软件许可协议。
- (2) 清查现有的IT资产清单。
- (3) 对比资产清单和购买记录，从而确定问题所在。
- (4) 卸载不合规的软件。
- (5) 针对用法和许可的合规性，实现管理策略。
- (6) 维护新的标准和过程。

365

SOA (Service Oriented Architecture, 面向服务的体系结构) 测试的目的是: 观察整个业务流程, 并确保流程的各个部分都正常交互。

端到端的SOA测试是指, 测试整个业务流程路径, 确保集成的结果是事务、交互和数据传输按预期执行。这还包括跨多个平台测试, 对传输协议、ESB、语言接口和消息进行测试, 确认业务服务和操作系统之间的连接和集成, 从而满足目标缺陷率和服务级协议 (SLA) 的要求。

随着SOA开始形成IT基础设施的结构, 积极灵活地测试Web服务已经变得至关重要。综合功能测试、性能测试、互操作性测试和漏洞测试构成了SOA测试的根本。

Web服务已经使网络设备、安全产品、应用以及企业中其他IT资产之间的界线变得很模糊。现在几乎每项IT资产都宣称自己的接口是为SOAP/XML消息传送准备的Web服务定义语言 (Web Services Definition Language, WSDL) 接口。在对企业内外集成领域, Web服务接口提供了前所未有的灵活性。这种灵活性使得所有领域的IT工作人员都必须负责确保Web服务在功能、性能、互操作性和安全需求方面和广告宣传中的一样好。

只有采用综合全面的测试委托书才能确保SOA健壮、可扩展、可互操作且安全。

SOA 测试的关键步骤

下面是每个成功的SOA测试策略都需要执行的步骤。

(1) 创建一个面向组装的计划。SOA和集成项目与传统的应用项目有着根本性的差异。在这些项目中, 大部分逻辑都用于处理各应用之间的连接, 而不是应用内部。从端到端的角度来协调和部署计划非常关键。要想让SOA项目不同凡响, 有好几种方法。首先, 需要一位有广阔的组织视野 (不能只局限于面向应用) 的项目/程序经理。其次, 做一个面向组装的计划, 融合传统代码和验证。但这个计划要以过程为中心, 并且以递增组装为导向。这其中最大的问题是, 大部分企业或公司仍然是以应用为单位或按垂直业务单元进行组织的。

(2) 关注需求和测试中的业务流程。关注业务流程, 听起来很简单。实际上, 测试一个业务流程意味着需要具备许多组件: 应用、中间件、支持技术以及支持这每一项的团队。因为流程横贯各应用和技术, 所以项目经理认为协调测试通常是最大的一个难题, 容易导致无法预测的延迟。

(3) 组建一个懂集成并且能够验证集成的测试团队。对业务重心的另一个挑战是要确保测试人员有广博的业务流程知识, 这包括理解错综复杂的业务事务中的“多米诺骨牌效应”。在多功

能团队中工作的能力，以及在知识获取和知识传递的文化背景下工作的能力，都是SOA测试人员的基本技能。组建一个由业务用户和测试人员组成的多职能团队，或者配备专门测试连接性的测试人员，这是成功的SOA团队创建以流程为中心的测试团队的两种途径。

(4) 在每个测试阶段对集成的连接性进行测试：单元测试、组件测试、集成测试和端到端测试。

理解了业务流程之后，测试人员就有能力访问底层技术来跟进和验证业务流程了。过去的质量保证团队需要开发人员编写数百个桩模块和harness，创建大量测试代码用于验证。但近几年出现了面向集成的测试工具，访问和测试SOA的底层连接性简单了许多。测试人员的重要转变不是仅仅关注代码，而是寻找有效方法来理解和诊断连接性。

多年来始终不变的是，集成团队总是在端到端的测试阶段努力组装系统中的所有组件，在这个测试阶段，能够最先找到集成问题的结果。为了避免易犯的集成错误，有必要使用在整个测试阶段通用的一致测试方式，在每个阶段测试SOA项目。SOA项目是组装项目，高效地测试SOA项目的唯一途径是一切从零开始。首先，在单元测试中，测试各个模块的输入和输出。然后将逻辑的每部分组织在一起，测试集成流的较小部分。这个新的测试阶段叫做“组装阶段”。这是关键的一步，等价于一次逻辑性组装一个Tinker Toy模型。只要完成这一步，就可以在全面组装的Tinker Toy模型上执行端到端测试。使用这一过程，端到端测试就真正是整个过程中最后一个确认，而不是调试集成逻辑的开始。

(5) 创建一个自动的可重复的测试过程。对于任何其他测试过程而言，可重复性都至关重要，尤其是在SOA项目中。系统路径的排列和组合数量庞大，是SOA项目中一项令人发憊的挑战。构建一个在有微小改变时也能正常运行的回归库，是以维护方式有效地减少集成级错误的唯一途径。虽然尽早测试一直是一项测试原则，因为这样容易找到错误，但SOA环境中微小错误的多米诺骨牌效应使得综合性的回归测试在SOA项目中必不可少。

(6) 针对典型的SOA测试障碍（如无法获得的系统）做计划。集成测试中一个反复出现的问题就是，并非所有需要测试的组件都可用。缺失的应用是一个内部模块？一个第三方应用？还是大型机系统的输出结果？“等待”造成集成项目中大量的时间浪费和协调延迟。模拟无法获得的系统的能力是保障SOA测试正常进行的必要条件。

无论你的SOA之旅刚刚起步，还是已经行进了一段路，一个面向SOA的测试策略都能显著改善SOA系统的交付，降低SOA成本。为了有效验证SOA系统，团队需要顺着SOA的路线思考，以组装和端到端测试为重心。不要把SOA测试想成单人短跑比赛，而应把它想成接力赛，注重传球。

368

369

30

软件开发生命周期中使用的方法论要么是迭代，要么遵循一定的模型，如瀑布模型。开发敏捷开发方法是为了对出现的变更作出快速响应，这些方法面向的是人而非过程。

敏捷方法论是一个集合，融合了值、原理以及结合迭代开发、测试和反馈形成新的应用开发方式的实践。迭代开发和敏捷开发提供了一种不同于传统瀑布方法的应用开发方法。

瀑布开发通过采用预先设计的方案来开发应用程序。敏捷开发则不然，敏捷开发是“沿途”开发需求。这意味着必须随着需求的演进开发测试用例。

31.1 敏捷用户故事与正式需求对比

IEEE (Institute of Electrical and Electronics Engineers, 美国电子电气工程师协会) 计算机协会发布了一套编写软件需求规约的指导原则。IEEE 的建议包括如何组织需求规约的文档、原型法的角色、好的需求的特性。IEEE 830 样式软件需求规约的最显著特征，是使用这样一种句式：“系统应该……。”这是 IEEE 推荐采用的编写功能需求的方式。

371

从敏捷开发的观点看，在编码之前就开发定义得很好的需求，有一些重大弊端。主要是这种方法意味着在某一时刻，软件的功能范围已经有了完整的定义，并被所有人接受。但实际情况并非如此。瀑布方法假设，软件满足一系列特定需求而不是满足目标用户的需求时，则认为软件是完整的。

31.2 什么是用户故事

用户故事 (user story) 是对需求的一种非正式陈述。在敏捷开发中，一个用户故事就是用用户的一两句日常用语描述的软件系统需求。用户故事用于表达需求规约和验收测试。每个故事都写在 3×5 英寸的小卡片上，因而不能写太长。

举一个用户故事的例子：当用户试图使用一张过期的信用卡时，系统提示他使用别的信用卡。

分解用户故事的过程很重要，因为这有助于思考如何开发和测试功能性。许多人把用户故事分解成几个任务来估算，因为这样它们就成了更小的工作单元，可以更准确地估算。分析用户故事的目的在于跟踪正在运行的被测试的特性的数量。我想知道有多少个用户故事通过测试。

31.3 敏捷计划

在敏捷计划中，我们经常要估算一个用户故事需要多少工作来完成。开发人员如果在设计和编码之前编写功能测试，能对用户故事的完整性更加自信。这对用户故事的估算也有帮助，能使发布计划更简单。

下面是一些有用的敏捷计划忠告。

(1) 做短期的进度安排。由于敏捷计划是一个演进的过程，所以很难做长期的计划。

(2) 不要过分强调甘特图。因为需求和活动都会快速地改变和发展，你可能会发现自己把所有时间都花在了更新甘特图上。

(3) 就进度安排与团队保持密切联系。团队应该积极参与项目，赢取支持。

(4) 定义短迭代。将开发迭代周期限制为2周到3周，也将限制“范围渐变”。

(5) 培训团队成员。团队成员可能以前从没做过敏捷项目，如果不参加培训和技能引导课程，就无法理解敏捷项目过程。

作为计划过程的一部分，为用户故事（针对所有需求）划分优先级不容小视，这可以通过图31-1所示的用户故事优先级模型来完成。

372

下面是用户故事优先级模型的基本元素。

- 关注某个用户故事时获得的相对收益。
- 忽略某个用户故事时的相对损失。
- 实现某个用户故事时的相对成本。
- 实现某个用户故事时的相对风险。

这其中的每个元素都是模型中所有用户故事的输入值，将用于权重计算。计算结果是一个数值，位于最后一列，这些值将按降序排序，表明它们的相对优先级。

31.4 敏捷测试的类型

测试驱动开发（Test-Driven Development, TTD）是一种软件开发技术，由短迭代构成。在这种迭代中，首先编写包含新功能的新测试用例，然后实现要通过测试所必需的生产代码。在实际开发之前进行测试能确保在发生任何变更后都能得到快速反馈。

31

相关从业人员常强调一点：测试驱动开发是一种设计软件的方法，而不仅仅是一种测试方法。测试驱动开发的第一步是将验收标准变成测试，因此对我们而言最重要的一点就是找到能帮助达到这个目的的工具。功能测试工具在测试过程中非常重要，几乎每一种编程语言都有多种单元测试工具，因此对测试人员而言，拥有一种合适的功能测试工具极其重要。

对于Web应用而言，可以使用Selenium、Watir（Watin、Watij）和Sahi等功能测试工具。Abbot可用于Java GUI应用，NUnitForm可用于Windows Form应用，Microsoft UIAutomation Framework可用于各种Windows应用。此外，也可以使用许多“重量级”的工具，如惠普的Quick Test Professional（QTP）。

准备实现用户故事之前，必须有一份由客户编写的正式验收测试，确保稍后能够判断故事目标是否已经达到。

用户故事优先级模型

用户指南

- (1) 输入列（从第1个用户故事开始）是相对收益（第C列）、相对损失（第D列）、相对成本（第G列）、相对风险（第I列），最低的为1，最高的为9。
- (2) 每项输入的权重可以调整（第10行，第C、D、G、I列），数字越大，权重越高。
- (3) 输入之后，从第13行开始，对第J列的值按降序排序，记录在第K列相应的单元格内，由此展现风险优先级。

相对权重		1		2		1		2		1		2	
		相对收益		相对损失		总 值		值的百分 比 (%)		相对成本		成本的百 分比 (%)	
总 数		0		0		0		0		0		0	
用户故事													
编号	用户故事												
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

图31-1 用户故事优先级模型

IT组织为了提升测试实践水平，经常将重心放在测试卓越中心（Center of Excellence, CoE）的某些甚至所有与测试相关的活动上。

图32-1描绘了测试卓越中心的一般组织结构。这个测试卓越中心主要由测试经理、测试架构师、解决方案架构师、测试自动化专家、质量保证经理和质量保证审核员组成。

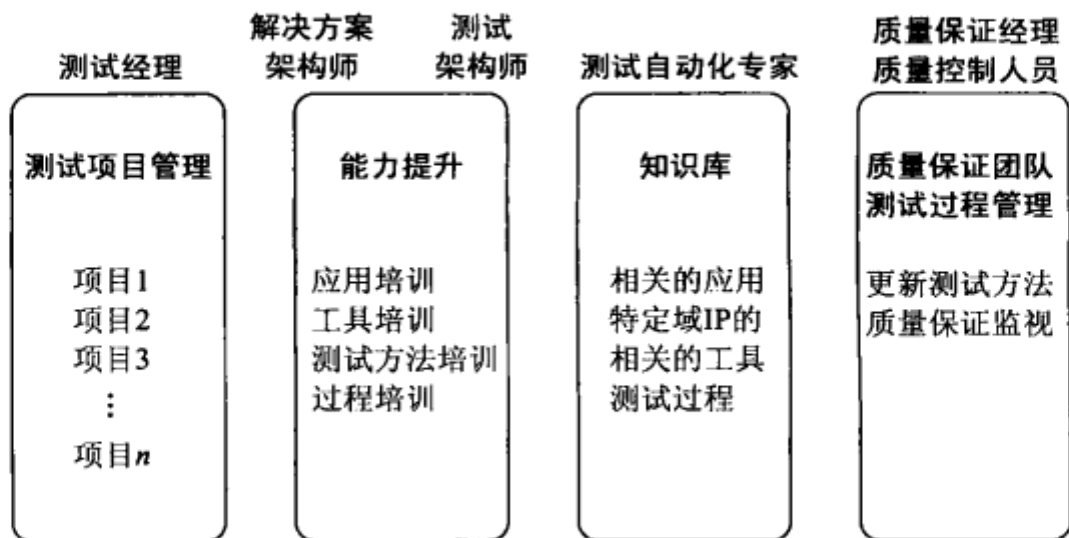


图32-1 卓越中心的组织结构

测试卓越中心致力于完成以下工作。

(1) 测试任务的分工。将多个项目在多个不同的领域处理。需要处理的测试任务可能有手工测试项目、测试自动化项目、性能测试项目，以及其他专项测试，如SOA测试、易用性测试、兼容性测试等。所有这些测试任务都由测试卓越中心分配处理。

(2) 能力的提升。这部分工作重点在于提高测试相关的能力，主要是安排各种培训，如应用培训、测试工具培训、测试方法培训等。

(3) 知识库的维护。这部分工作就是为所有的测试项目维护知识库。图32-1中第四部分由具备质量保证能力的人组成，他们将负责质量审计和审查。

应用开发的业务视角正在发生着显著变化，企业用户逐渐变成应用开发和应用部署的主要驱动因素。

独立的软件测试一出现，就成了信息技术学校学生主修课程的关键业务，企业也开始成立专门的测试中心来应对日益增加的测试需求。将工具、技术与面向过程的综合测试方法集成，是这

些服务提供者的期望。

测试卓越中心应该提供以下要素。

- 端到端的测试管理过程，包括测试过程和阶段方法论。
- 不同阶段针对验证和确认的评审过程。
- 进入标准。
- 质量保证测试过程的退出标准。
- 缺陷管理。
- 自动化测试工具实验室，包括测试自动化的标准。
- 发布管理。
- 根源分析。
- 测试报告和沟通。
- 测试阶段的定量测试过程管理（通过特定的度量和测量）。
- 测试相关的制品和其他可复用组件的知识管理系统。

下面介绍建立测试卓越中心的步骤并描述测试服务趋势和测试卓越中心的促成者。

(1) 建立一个新的测试卓越中心。这是一个迭代过程，对于根据你所在组织的实际需要来理解和建立卓越中心非常重要，因为需求往往是特定于具体公司的。建议遵循下文描述的步骤，它们体现了持续的评估和改善。

378

(2) 定义预期目标。测试卓越中心的建立应该符合组织的业务目标。应该与组织的主要利益相关人磋商以后再定义卓越中心的范围。这个阶段将提供测试团队运作的框架。

在这个阶段，解决方案架构师将了解组织中的质量管理过程及其成熟度，这有助于确定测试中心的范围。确立测试卓越中心的框架时，我们需要确保满足项目发起人的期望和项目目标。

除了项目发起人和主要利益相关人之外，还需要满足业务和IT系统中其他关键角色的要求，以巩固跨业务线的范围。与主要利益相关人达成一致是建立测试中心的关键。应用负责人、业务单元负责人、IS管理团队和业务技术主管也属于关键人物，因为他们决定着卓越中心的结构和与其他业务单元之间的沟通通道。

(3) 定义当前的测试模型。每一项改善都应该来自于已定义的标准模型。有必要分析和记录现有的测试过程，以便将来度量测试中心的效益。在这个阶段，团队需要访问IS管理层、开发经理、质量保证团队、业务技术经理和用户代表。

解决方案架构师将与现有的质量保证资源打交道，了解他们的一些信息：

- 过程知识——项目管理、SDLC和测试过程；
- 业务知识——每个产品线的领域知识或功能性知识；
- 应用或项目知识——项目需求、过程和技术细节，包括特定测试项目的相关知识，如年度维护的发布、补丁的发布和相关基础设施。

这一阶段将帮助测试人员了解现有项目的渗透度和过程相关的知识传递方案，以及它们的有效性；帮助建立交叉培训技术，建立有助于风险管理和保持期管理的旋转计划。

在此阶段的末尾，解决方案架构师将完成一份范围文档，给出卓越中心的结构、已定义的角色和相关负责人的职责。这一文档还会提出卓越中心的长期目标，说明将在各个业务线上引入怎

379

样的一致质量过程来平衡组织的业务发展。这个阶段将给定当前标准下的一系列定量测试过程的度量，以及如何在各时间段提高它们。

(4) 定义测试卓越中心的范围。必须定义测试卓越中心的初始范围。由于可能会出现许多全局修改，所以最好一次只实现一步，便于分辨测试卓越中心产生最大业务影响的地方。这不能一劳永逸地取代全部的质量保证工作，因而应该关注特殊的业务单元或应用类型，或者特定的应用。采用这种定义方法，初期投资、初期策略和流程可以在分支到其他区域之前得到验证。度量也将在这阶段定义，用以评估测试卓越中心的成功。

(5) 开发实现计划。确定范围之后，还需要制定许多具体的计划来正确地资助测试卓越中心。开发实现计划包括开发基础设施需求、工具集和计划人员配给，还包括定义标准、策略和卓越中心建立到位后要实施的规程，此外，还会确定统一的测试模板、原则和检查表。

这一阶段的关键是找出所有内在风险，以确保最终的运营模型能满足所有潜在的约束条件。测试人员需要用文档记录对当前测试工作流有风险的因素、已评审过的项目的详细的积极元素以及所有会降低项目风险的建议。

(6) 建立测试卓越中心的服务。这一阶段将逐步实现之前定义的计划，购买和部署基础设施与工具集，确定卓越中心的地址，然后开始测试现有计划。在这一阶段的最后，业务线（line of business, LOB）将能够使用测试卓越中心的服务来保证质量。下面是这个实现阶段用到的主要组件。

- 确定卓越中心的资源和测试项目。
- 达成分阶段实现的业务优先级协议。
- 识别可能的实现项目限制和依赖关系。
- 建议实现的风险分析和风险缓解的定义。
- 建立优先级，确定候选项目。

(7) 评估成果。在这一阶段评估之前定义的度量和服务的整体客户满意度。测试卓越中心任何一方面需要作出的调整，都将以度量和客户满意度为基础来完成。一旦作出这些调整，测试卓越中心的范围就扩大了，并将这项服务提供给公司里一个更大的团队。这一过程的目标是以公司的需要为基础，实现卓越中心的扩大，直到中心的收益福泽整个公司。

(8) 结构和最佳实践。测试卓越中心经理领导着整个中心，除了管理测试项目的各种指定测试之外，他还负责启动各种核心技术活动。图32-1描述了一个测试卓越中心的一些关键活动。持续过程改进和在测试技术中引入新方法以减少测试时间和提高投资回报率，是测试卓越中心经理的主要绩效目标。

(9) 测试方法。测试卓越中心采用指定的ETVX标准（Entry、Task、Validation、Exit）创建特定的应用/产品测试和实现方法，创建标准化的模板、标准、原则和检查表，建立同行评审和领导评审过程，制定全面的项目管理计划，包括工作量评估、资源管理计划、风险鉴别和风险缓解计划、假设条件以及每个测试阶段的进入标准和退出标准。

32.1 行业最佳过程

测试卓越中心采用的都是行业最佳过程，这些测试过程被很好地调整为IEEE/ISO/CMM/

TMM过程。

TQM负责针对质量概念培训所有的人力资源，卓越中心将采用这些最佳过程来向客户交付高质量服务。

32.2 测试度量

图32-2所示是卓越中心将会采用的一套测试度量集合。

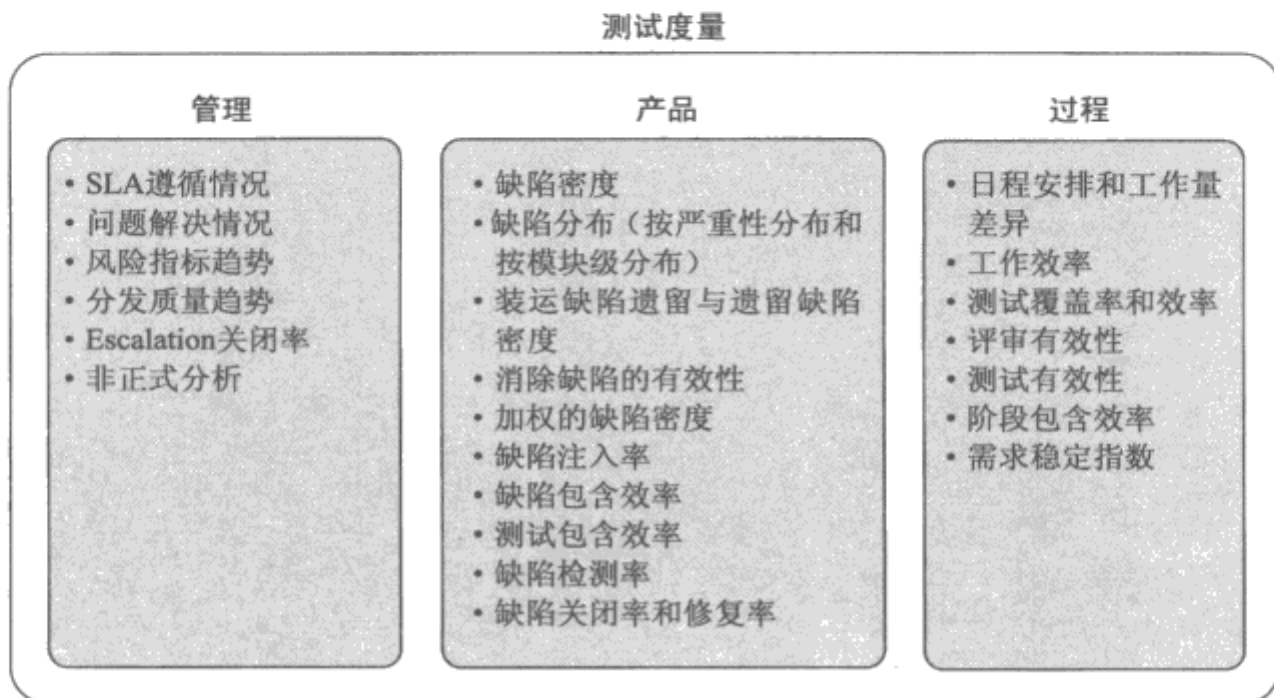


图32-2 测试度量

32.3 运营模型

测试卓越中心经理看重灵敏度、高效性、可靠性和瞬时可扩展性，有人力资源管理才能，在运营模型中具备核心管理能力，并且工作主动。他负责规定架构、测试和ASQ工具标准，具备丰富的业务模型知识，是整个卓越中心的支柱角色。

解决方案架构师负责定义整体测试策略的解决方案，定义卓越中心的运营模型、发展模型和盈利模型。

测试架构师是卓越中心另一个关键角色，负责大部分咨询工作，拥有深入的针对特定的纵向需求的测试过程知识，负责对过程的定义、改善和持续提升，并且根据业务度量对性能进行测量。

自动化专家负责协调技术和ASQ工具的业务应用级别的所有活动，如测试设计、脚本开发、定制、执行和集成等。

32.4 测试自动化框架

卓越中心会创建一个测试自动化框架，在此框架下创建所有的功能性回归测试套件。这个框

架将定义可复用的业务功能、自动化构件、目录结构，以及编写自动化脚本的标准和准则，这使得客户可以充分利用有效的可复用性和自动化增强包。

卓越中心还会培训测试人员采用功能自动化工具和性能测试工具。

32.5 能力的持续提升

卓越中心鼓励组织中的人获取质量相关领域的证书，如质量保证协会颁发的注册软件测试工程师认证（QAI CSTE）、注册软件质量分析师认证（CSQA）、注册软件项目经理认证（CSPM）以及工具厂商颁发的注册产品专家认证。测试卓越中心还会创建一个全面的年度能力提升计划，不断提高人员的测试技能和其他能力。

在20世纪，外包已经成为一个重要的概念，这个时期许许多多的制造业产品被外包到中国和日本进行生产。离岸外包模型可以为客户提供额外价值。效率的提升和成本优势是外包产业增长的两个基本因素。当IT泡沫破灭，许多公司都不得不为了保持自己的利润基线而努力的时候，通常会把开发、测试和维护外包到亚洲国家，如印度、中国等。本章分析了关于测试外包的重要要素、相关优势和所涉及的一些问题。

根据美国质量协会（American Society for Quality, ASQ）的定义，项目管理是指运用各种知识、技能、工具和技术满足项目需求的过程。项目管理知识和实践最好是描述为一系列过程。关键项目管理过程是软件测试。下面几节描述管理这一行为的实现考虑因素。

本部分的目标如下：

- 描述基本的项目管理原则；
- 对比一般的项目管理和测试管理；
- 描述如何有效地评估测试项目；
- 描述缺陷管理子过程；
- 讨论离岸和在岸模型的优缺点；
- 列出将测试活动集成到开发方法中的步骤。

383

33.1 步骤 1：分析

公司管理部门应当对外包选择进行估算。外包估算不应当作为例行的活动。要做出外包的决策，应当有一个委员会对以下方面进行仔细研究，并向高层管理者提交一份报告。

下面是一些需要回答的问题。

- 产品、开发、测试或维护可以外包吗？
- 项目的哪个部分可以外包？
- 关于外包，其他公司有怎样的经验？
- 市场中的产品线是否有可用的选择？
- 将要外包的短期和长期的业务流程如何？
- 需求是否预先定义好了？
- 外包项目的业务流程变动是否频繁？

3

□ 接包方所在地是否有所需的额外能力？

管理层应该对报告加以分析，只有确定通过产品外包可以给公司带来丰厚利润的时候，才能做出产品外包的决策。

33.2 步骤 2：确定经济上的得失

一旦管理层决定了要将产品外包，就应当对其成本和收益进行分析。这可以从以下两个方面进行分析：

- (1) 在同一个地理区域内，外包在成本上的优势；
- (2) 在有多个可选的外包接包方竞争时，哪个更具优势。

不能仅凭接包方做出的成本-收益分析进行合作判定，这些分析的最终目的是为了获得新的商业项目，并没有考虑到项目操作中通常应该包含的返工和意料外项目成本。80%的成本节省行为都将外包项目包给了无法按时提供高质量产品的接包方手中。研究表明，外包公司一般都有40%~50%的成本收益，任何不符合这一行业平均值的项目预算都是值得怀疑的。

384

33.3 步骤 3：确定选择标准

在美国的许多公司里，外包项目已经成为了他们的灾难，原因很简单，他们选择外包公司的过程是错误的。

- 地理位置：外包国家的位置是选择外包合作伙伴的一个主要因素。由于其地理位置，像印度和中国这样的国家获得了更多的外包项目，因为在这些地方可以确保7×24小时的工作制。当美国的公司早上开始工作时，前一天送到这些地方的需求已经完成了两个轮班。项目管理组经常发现很难评估这天早上收到的输出。
- 最优使用：如果在不同的地理位置进行7×24小时工作的话，就可以达到对硬件资源的最大程度使用。由于全球网络都可以通过IPLC和VPN网连接起来，这样对服务器的访问就可以被分配到全球不同的地理位置上去，从而确保了各地服务器的最大程度使用。
- 高质量的交付物：应当按照本公司的质量标准和国际质量标准评估接包方交付物的质量。

33.4 项目管理和监控

本土/离岸模型的成功与否取决于公司是否采用了有效的项目管理手段。很多使用这个模型进行实验的公司已经开始放弃这个模型。除非为本土/离岸团队建立了明确的管理方法，其中定义了明确的角色任务和职责，否则这个模型必然要在交付预期的成本优势时遇到一些障碍。

33.5 外包方法

决定外包之后，外包方法应明确地定义什么能外包，什么不能外包。可以外包的活动如图33-1所示。当然，这些活动严重依赖于需求的完整性、沟通效率、项目支持的执行效率和策略以及要有定义良好的项目管理、质量保证和开发过程。

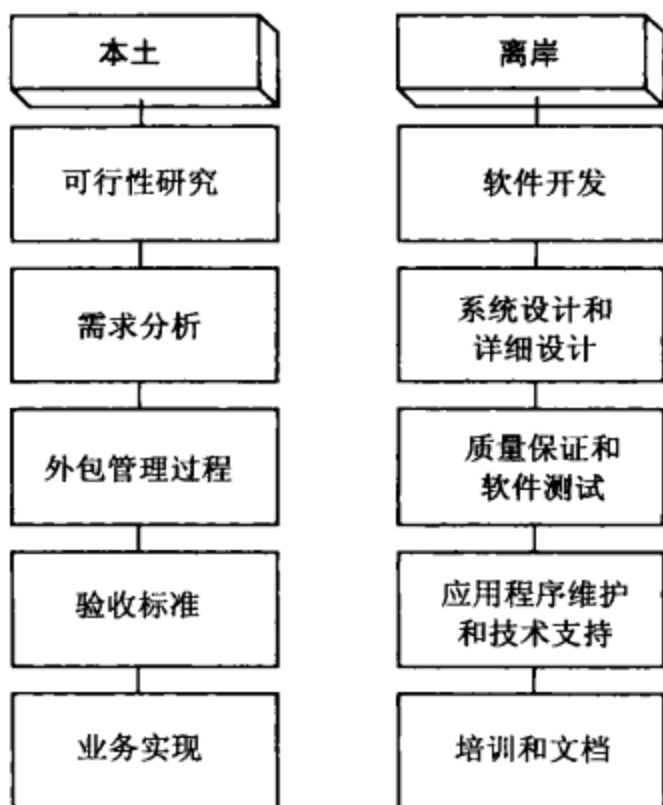


图33-1 本土活动与离岸活动

工作成本越低、工作速度越快，执行效率就会越高，两者是密切相关的。策略则与建立一个长期业务竞争优势相关。如果一个应用程序开发过程具有很重要的战略意义，而且有多个螺旋迭代过程，则不能把整个开发活动和质量保证活动外包。一个需求定义得非常好并且对业务运作过程没有什么本质影响的应用程序开发项目，可以作为离岸外包的候选。

33.5.1 本土活动

- 企业应该对特定的开发/测试/维护活动的外包进行最初的可行性研究，以确定这些活动是否可以外包，而这一部分活动显然是不能外包的。如上文所述，这一研究结果应该包括成本-收益分析和外包的优劣势分析的内容。
- 在决定部分或整体过程外包之后，应该进一步进行需求分析，以决定将要外包的业务过程的比例。这些需求应当明确地记录在文档中，以使在管理层和接包方之间在交付物和预期成果的沟通上不会存在隔阂。
- 为了有效地监控和管理外包项目，应当建立项目管理程序。应当将本土协调部门和离岸团队的任务角色和职责明确地记录在文档中。以上都应由管理层决定，并由管理高层和接包方同时签字认可。
- 外包项目的验收标准应明确写入文档。应由公司通过引入实际最终用户或业务分析顾问进行验收测试。应为验收标准建立适当的指导方针和检查表。
- 应由本公司进行业务实现活动，因为这是不能外包的。

33.5.2 离岸活动

下面列举的是可以外包的活动。

- 开发——就全球而言，软件开发过程是目前最主要的外包活动。许多软件公司都是集中分布在印度、中国和东南亚这样的地区，这些地区在软件代码开发方面对英国和美国给予了非常有效的支持。这些公司大都通过了如软件能力成熟度模型（CMM）、国际标准化组织（ISO）和其他国际审计标准的认证，并且为检验交付物建立了清晰的流程。
- 总体设计和详细设计——随着这些国家英语知识的普及和全球网络一体化，除了软件外包之外，其他相关活动如总体设计和系统设计实际也可以被离岸外包。
- 系统测试——随着全球沟通方式的简单化和低成本，许多测试活动也可以外包。由于系统设计和开发都在离岸开发中心进行，系统测试当然也可以离岸外包。这能减少雇用额外高级人才的费用，也不会打乱现有业务资源的日常分配而间接影响业务增长。
- 质量保证——质量保证和软件测试是美国公司可以选择外包的另外两个重要活动，这与开发外包和其他活动直接相关。大部分接包方都有现代软件测试工具开发的专业技术，他们可以在全球范围内任何地方执行这些自动化测试脚本。
- 支持和维护——应用的维护和支持（AMS）工作是可以进行长期外包的另一个关键活动。许多企业（如英国铁路和主要电信公司）关键应用的呼叫中心都已经移到了印度之类的国家。
- 后续活动——在给出交付物标准的条件下，任何与软件开发、设计或质量保证相关的文档工作都可以外包。随着互联网工具（如Placeware）的开发，甚至连培训都可以外包到遥远的地方。

387

33.6 实现本土/离岸模型

一旦在最初的分析阶段之后决定外包，外包过程就应当实施分阶段的管理。阶段的划分不应该影响公司现有的业务活动。以下5个阶段定义了这个过程。

33.6.1 知识转移

在这个阶段，离岸核心团队可以拜访客户以了解需要外包的应用程序。把这些技术和业务资源与现有资源进行融合，并将应用程序的功能、技术和操作等方面进行消化。一般来说，要任命一个客户协调人员作为联络员进行计划、监控和评价知识交流活动。离岸团队要做出反馈，表示其对系统已经理解，以使客户协调员能确信他们已经掌握必要的知识。这个团队要准备对其他离岸团队人员的培训，以及给他们提供文档。

这个团队应由外包公司的特定专家组成，这些人会把客户所要求的后续过程记录成文，这些过程将与接包方过程整合或者联合在一起，以使得接包方的交付物能够通过审计要求。

33.6.2 详细设计

一旦完成了最初的知识获取阶段，技术团队将为硬件需求、软件需求和联络需求进行一个详细设计，只有详细设计完成后才能开始远程运作。客户方的技术团队要对这个设计加以验证。一旦技术的详细设计获得了批准，远程的基础设施团队将开始搭建工作环境。客户方的服务器和应用程序将彼此联通，并通过健全性测试。

业务分析团队要准备一个转移计划，分阶段地转移计划中的活动，如开发、测试、设计或维护等。

33.6.3 基于里程碑的转移

本土/离岸模型的转移过程提供了按步骤地将开发、测试、设计、支持和维护的职责从发包方转移到离岸方的框架，而不影响客户的正常业务。要进行平滑的转移，关键的里程碑包括：

388

- 在离岸方建立工作环境；
- 培训离岸人力资源；
- 计划分阶段转移的确定的活动。
- 获得离岸稳定性。

33.6.4 稳定状态

在一段时间之后，离岸环境将稳定下来，高质量而低成本的交付物将开始提交给发包者。这是一个关键时期，这个时期本土项目管理活动应小心翼翼地检查交付物，并安排与离岸团队或其他相关活动的会议，以澄清其中的问题。一旦到了这个稳定状态，模型就算已经建立起来了。

33.6.5 应用管理

一旦设计、开发和测试都已完成，产品已上线，这时产品可能需要进一步的增强（比如因为新的需求需要修改代码），进而需要维护。此外，在正常的业务周期中，例如数据备份和净化数据等的工作也可以外包。因为接包方公司在这些业务领域具有专长，他们能长期低价地提供这些服务。

理想的话，20%~30%的工作是本土完成的，而70%~80%的工作是离岸外包完成的，这个比例取决于项目的重要程度。一般来说，需求分析、详细需求的开发、关键支持和实现都是在本土进行的，开发和测试是离岸外包进行的。

33.7 先决条件

下面是一些本土/离岸模型能够有效得到预想结果的重要先决条件。

33.7.1 关系模型

要得到一个成功的本土/离岸模型，应该建立关系模型。图33-2给出了概括的关系模型。本土项目经理、本土协调员和离岸团队的任务角色和职责应有明确定义。

389

下面是本土客户团队的一些一般职责。

- 发起并参加所有状态的会议。
- 协调并提供关于测试需求的信息。
- 对离岸团队提供问题解释和向其提供其他所需数据。
- 评审离岸测试交付物，并对交付物的质量签字认可。
- 与离岸项目经理沟通，从各自的本土部门获取对问题的解释和对测试交付物的批复。
- 为问题澄清和交付物建立并确保最佳周转时间。

33

- 批准时间表。
- 与离岸项目经理磋商，决定测试项目的时间线/进度表。
- 为各方准备并发布所有测试请求要点的日常状态。
- 主动通知对离岸团队交付物有影响的请求或其他变更。

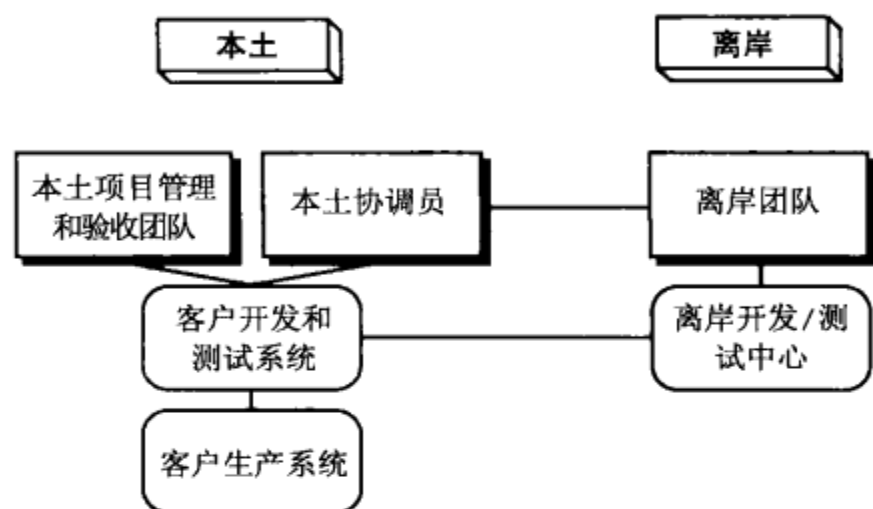


图33-2 关系模型

下面是本土客户团队的一些一般职责。

- 指定与本土协调人员互动时对离岸团队的单线联系。
- 决定每日/每周/每月状态评审会议的时间和方式，这些会议的参与者应当根据利益相关人不同的级别来确定。
- 定义整个项目管理活动和评审过程。
- 确定每周/每月状态报告的格式和内容。
- 通过状态报告准备和发布每日进度。
- 与本土协调人员沟通以澄清开发/测试团队工程师提出的问题。
- 支持和评审开发与测试团队工程师准备的所有项目相关的测试文档和交付物。
- 将测试项目分配给离岸测试工程师。
- 根据项目的不同阶段的项目需求确认项目所需资源的增减。
- 准备项目计划和策略文档。
- 主动通知本土协调人员项目相关问题。
- 明确告知交付物质量的职责。
- 决定项目跟踪和项目控制的职责。
- 进行每日和每周团队会议。
- 收集时间表，并提交给本土协调员。
- 在与本土协调员协商时，为测试项目最后决定时间线/进度表。

33.7.2 标准

很多公司试图采用一些最节省成本的本土/离岸模型最终失败了，主要归因于发包公司和接

包方公司在技术标准和指导方针上的差异。尽管双方公司都通过了CMM和ISO认证,但是他们针对同样的源代码或者测试方法采用的标准还是可能有巨大的区别。

应当使用发包公司的标准和指导方针评估和同步接包方公司的标准。开发标准、测试方法、测试自动化标准、文档标准和培训范围都应预先加以明确定义。除了以上这些,下面的问题也应加以考虑。

- 征求意见(RFP)规程。
- 变更请求规程。
- 配置管理。
- 工具。
- 验收标准。

验收标准是应当在使用该系统的最终用户的帮助下定义的另外一个重要标准因素。如果用户不能接受该标准和交付物,那么这个项目实现阶段就失败了。

391

33.8 本土/离岸方法的收益

这一模型的最终优势是节省时间、金钱和沟通。图33-3中描述的案例研究给出了离岸模型使用的理想情况,并描述了公司可能从本土/离岸中所获得的大致收益。这个统计数字从80%~40%不等。

编号	描 述	比例	天数	人数	总数
					(以美元计的总数)
1	本土: 5人6个月	640	132	5	422 400
	项目管理工作量, 总成本的30%				126 720
	总数				549 120
2	离岸: 5人6个月	150	132	5	99 000
	项目管理工作量, 总成本的30%				29 700
	本土的知识转移				25 000
	最初的网络连通				50 000
	重复发生的成本(维护)				5 000
	沟通开销				5 000
	管理开销				5 000
	总数				218 700
	分析:				
	本土和离岸之间的差别				330 420
	离岸模型对本土模型收益百分比				60.17

图33-3 成本-收益案例研究: 本土和离岸模型之间的成本-收益分析

392

在案例研究中, XYZ公司将在分析中估算以下变量(关键成功因素在变量10~13中标出):

- (1) 所需人力资源的数目——5人。
- (2) 项目日程设假为6个月。
- (3) 本土已有的人力资源成本假设为80美元/小时。
- (4) 考虑每天8小时, 一个月22天。
- (5) 假设离岸人力资源成本是150美元/天。
- (6) 假设初始知识转移大概需要用两周的名义成本。
- (7) 项目管理成本按照人力资源成本的30%估算。
- (8) 初始环境的建立成本和后续维护成本都按正常情况假设。
- (9) 包括重复发生管理成本的百分比。
- (10) 需求经过完整地评审, 并由利益相关人签字认可。
- (11) 项目经理建立了沟通渠道, 运行良好。
- (12) 项目对业务而言是非战略性的。
- (13) 建立并记录了项目完成的标准。

尽管对于西方国家来说, 要满足预期的对IT人力资源的需求是极其困难的, 但是在中国和印度等国家的实际情况恰恰相反, 那里有许多具有良好教育背景的优秀程序员。一般来说, 这些接包方公司都拥有一个广大的、高度专业化的知识基础。

另外一个优势就是我们可以仅在需要的时候才去雇用这些人, 而不是连续雇用他们。这可以尽量地帮助公司节约成本。

由于这些地区的地理位置优势, 可以从接包方获得7×24小时的服务。在美国一天结束时发出的服务需求在第二天一早就可获得交付物。

大部分外包公司都通过了CMM五级, 并有ISO认证, 他们具有确定的工作流程和规程, 这大大提高了交付物的质量。

本土/离岸模型可以使商业组织集中精力在他们的核心业务上, 进行业务重组, 为协助顶层管理层制定决策和给中低层的成本和质量控制提供有效、及时和充足的信息。

本土/离岸模型的挑战

下面是面临的主要挑战。

1. 不在可控视野内

离岸团队处于视线之外, 这增加了本土项目经理对失控问题的担忧。这个问题可以通过访问接包方利益相关人和接包方离岸设施而获得信心来解决。此外, 因为接包方采用了CMM、ISO和IEEE认证等工业标准, 确立的过程、方法和工具也可以为本土方提供信心。

2. 建立透明性

接包方应向客户提供完全的透明性, 并允许他们积极参与离岸人力资源的招募, 并在他们认为合适的时候启用他们自己的本土人力资源。

3. 安全性考虑

数据保密的安全性考虑可以通过一个用户专用的网络克服。

393

4. 项目监控

如果需要通过裁减项目管理实践活动以满足客户的需求，就已经接近项目管理失败的边缘了。

5. 企业管理人员成本

当估算整体成本收益时，额外的管理人员费用是必须计算在内的，并且使用这个模型的花销可能是很大的。

6. 文化差异

尽管对于从欧洲和美国外包的多数项目来说，掌握流利的英语可以被看做一种优势，但是文化的差异可能会制造一些困难。不过，个人一般都能很快地融入不同的文化环境。

7. 软件许可证

软件许可证使用是与全球许可和地区限制相关的又一个问题，需要具体情况具体分析。

33.9 本土/离岸模型的未来

一些公司已经尝试或正在尝试本土/离岸模型以减少IT成本。尽管成本的节省是很明显的，但是离岸交付物的质量很大程度上取决于明确的本土项目管理方法和标准。尽管由于全球时间差异的交迭有每周7天每天24小时的范例，但是沟通和文化调整也是成功的关键因素。

394

然而，如果技术公司还没搞明白运营效率和策略就选择离岸外包将是一个错误的选择。运营效率与工作的低成本和高速度相关，而策略则与建立长期的竞争优势有关，这对于技术公司来说，通常是指软件的创新力。

如果开发的软件不是公司实际销售的产品的创新渠道的重要环节，选择外包开发或质量保证测试是可行的。例如，Web站点设计或后台管理软件，诸如账户支付或存货控制软件都可以被外包，这是一个有效的方法，因为它提高了运营效率。但是编写和测试创新的软件是不能在装配线上生产的，这需要一些很难的开发、设计和测试技巧，所以选择外包开发工作并不具备竞争优势。一旦一个技术型公司失去了创新能力，它也就走到了丧失竞争力的边缘了。

395

现代软件测试工具

测试自动化会把测试团队的工作复杂化，并且影响成本。但是，合适的人经过正确的培训在适宜的环境和适合的条件下同样可以使其提供有价值的帮助。

自动化测试软件的繁荣发展得益于基于Web方式的、C/S架构的和大型机的软件开发工具，这些工具使得开发人员可以使用日益强大的功能来快速开发应用程序。测试这一学科必须面对日益完善却更加复杂化的软件。新的用于质量保证过程的测试工具已经开发出来。

本部分的目标如下：

- 描述未来的测试工具；
- 测试工具的使用条件；
- 什么时候不需要使用测试工具；
- 提供测试工具选择检查表；
- 讨论测试工具的类型；
- 对现代流行的测试工具进行说明；
- 阐述评价测试工具的方法。

今天许多公司仍然对需求阶段感到很头疼，需求阶段时常被缩减到（或者省略掉）能让应用程序“出门”（发布）的最低标准。测试人员经常在问：“如何测试没有需求规约的软件？”答案是：无法测试。缺乏好的需求定义会因为连锁效应而造成每年数十亿美元的损失，如果一个开发生命周期阶段还没有彻底完成就开始着手进行到下一个阶段，连锁效应就要发生了。举例来说，如果需求没有被完全定义，在设计和编码时就可能会反映出错误的需求。应用程序开发项目经常会不断地重新定义需求，然后重新设计并重新编码。第四代编程语言的高效使得软件开发的学习简单了许多。

但是，软件测试也遵循上述的软件开发的历史趋势，如下所述。

34.1 自动捕获/回放测试工具

自动化测试工具最初的目的是自动回归测试，确定软件的变更不会影响已经测试过的应用程序。这需要测试人员设计出一些可重复执行的详细测试用例，而且在每一次应用程序发生变更之后都执行对应的测试系列。随着自动化测试工具的普及，很多人把自动化测试工具当成解放测试苦力者的希望。

399

然而，所有的测试工具与测试过程或方法都是相关的。测试过程由测试计划、测试设计、测试实现、测试执行和缺陷管理组成。自动化测试工具必须与测试过程相结合，而测试过程必须与开发方法相结合才能获得成功。仅有一些测试过程是不够的，很多公司在过程改造的同时衰落或者失败了。他们赢得了波多里奇奖并且创造了惊人的效率、低成本以及产品质量与客户服务质量的改进。公司经历这样一个悖论后，清楚地发现了正确的过程，但是这和使正确的过程正确并不一样。选择和使用自动化测试工具并不能保证一定成功。

34.2 测试用例构建工具

公司购买了许多自动化测试用工具，但他们很快意识到他们需要有编程经验的程序员或测试人员来创建与维护脚本。重点变成了如何使自动脚本（一个程序）正常工作。编写脚本是开发项目中的一项开发项目，需要编写很多的程序。但是，许多测试人员没有编程背景，同时开发人员又不愿意做测试。

自动化测试工具只是把测试数据传递给要测试的目标应用程序。自动化测试工具一般用于功

能/GUI测试。工具是测试数据和图形用户界面之间的接口，用来验证目标应用程序的响应符合需求中的定义（如果有的话）。测试数据/场景编写是一个由测试人员（或业务分析人员）完成的将需求（通常用文字处理工具编写，如微软的Word）转换为测试数据的手工过程。

这是一个非常耗时的、困难的问题，由手工完成的效率很低。有很多的测试技术（参见附录G）可以帮助进行这种转换，但是这种从一种格式到另外一种格式的转换始终是一个手工工作，比如，将一段英文陈述转换为测试数据/场景。具有讽刺意味的是，很多人都非常注意测试场景的开发，却很少有人关心测试数据的质量。

34.3 必要条件和充分条件

当今的自动化测试工具并不足以保证测试的质量，这些工具充其量也就是和从自动化测试脚本中输入的测试数据差不多。

GIGO的意思是输入/输出全是垃圾（garbage in, garbage out）。电脑与人不同，即使大部分的输入数据毫无意义，它照样会忠实执行，而且还会产生出无意义的输出。当然，好的程序应当拒绝处理明显错误的输入数据，但是这种检查机制一般来讲是很不容易定义和实现的。那些输入了垃圾数据的人经常抱怨说程序没有“做正确的事情”，这就是所谓的GIGO了。这个术语也普遍用来描述由于错误的、不完全的或不严密的数据而造成的人为的错误决策。人类不得不全部信赖“计算机处理”的数据，这真是讽刺啊！

保证质量的充分和必要条件是有一个健壮的工具作为测试数据和测试场景的传递者，该工具将根据应用程序的功能选择合适的测试数据和测试场景对目标应用程序进行测试。对于绝大多数的商业应用程序而言，测试数据是决定测试结果的关键。不仅要输入和验证数据值，同时还要知道数据将要达到的状态，这样才能够预计测试结果。掌握测试数据是任何测试活动的基础，因为软件测试的一个基本原则就是必须知道数据的输入条件和期望的输出结果，只有这样才能进行有效的测试。如果这些都不知道，那么这就不能叫做测试了，充其量不过是一次实验，因为不知道会发生什么。可预测性对手工测试是很重要的，但是对于自动化测试则是必要的。

但是对于大多数系统，必须准备好足够的有意义的数据才可以开始测试，如果需要成千上万甚至更多的数据记录，这就成为了一个新的问题。在极端情况下，测试一个航空票务程序需要初始化成千上万个城市、航班、旅客和费用来满足执行所有的需求。真正的测试本身倒没有准备数据所花的时间多。人员和过程是测试的另外一个必要条件，我们需要培训出合格的测试人员，还有在尝试测试自动化之前必须要有一个稳定的测试过程。

34.4 测试数据生成策略

历史上有4种整合测试数据环境的基本策略：生产数据抽样、从零开始、数据播种或从数据库生成。每种策略都有各自的优缺点。在讨论完这4种策略之后，我们将讨论第5种策略，它不仅产生测试数据同时还产生测试场景和期望结果（基于需求）。

34.4.1 生产数据抽样

最常见的获得测试数据的方法是从生产数据中获取。这种方法看起来既合乎逻辑又具备可实

践性：生产数据代表了现实，包含了软件必须处理的实际情况，提供了问题的深度和广度，同时很明显可以在很短的时间内创建大量的数据。

401

然而，这种方法至少有3个主要的缺点。很少有测试平台能够完全具有与生产环境相同的处理能力，因此必须提取出一个可以供实际测试用的数据子集。这个子集的获得并不是简单的隔 N 条记录选取1条或者统一地选择数据的一个百分比。文件之间复杂的相互关系决定了我们选取的子集也必须是内聚的。举例来说，所选择的交易必须和有效的主账户相关联，而且合计数必须和账户余额还有历史交易明细相一致。单是标识所选子集本身的相互关系和保证子集有意义就已经是非常繁琐的任务了，我们很难知道到底要多大的子集才能实现对主要状态和组合的覆盖。

这种方法的第二个主要缺点在于必须不断地更改测试本身和提取的数据，才能够使得它们能够一起工作。回到我们的基本原则，我们必须知道一个有效的测试的输入条件，在这里就是数据的内容。每次提取新的数据都会让所有事情重来一遍。比如，我们进行一个所得税的测试，要求有一个雇员一年至当前日期的收入超过FICA对下一次薪水的限额，执行测试的人员必须在子集中找到这样的一个雇员进行修改或添加一个。如果是自动化测试必须更改这个新雇员的编号及相关资料。搜索符合你提出的所有条件的雇员，就如同大海捞针一样困难。因此，由于有限的可重复性，节省时间的想法就化成泡影了。每次更新数据后都得重新设置测试条件。

最后，我们可以看到这种方法明显不能用于正在开发的新系统，因为根本没有产品数据可用。

34.4.2 从零开始

另一个极端是从零开始，实际上就是每次都重新生成测试数据。这种方法的好处在于能够完全控制，内容也总是可获得的，而且随着时间可以增强或扩展，这样就保留了以前的工作。由软件本身创建并维护相互关系保证了内聚性，文件结构的变更或者记录的布局能够被自动关联起来。

但是测试数据的重构是有风险的。显而易见，如果没有自动化，对于大规模的应用程序这是极不现实的。其次，一些文件是不能通过在线交互来创建的，他们只能由系统通过接口或者处理周期来生成。因此，可能无法从一个真正干净的状态开始。

另外值得注意的是，凭空制造的数据针对产品的扩展性比较差：在实际使用中经常出现的一些特定和特殊的情况可能没有被测试设计者考虑到。所以，这种方法必须对测试数据进行稳定的和不断的扩充，但由于缺乏随机性而掩盖了产品的很多问题。

34.4.3 数据播种

402

数据播种是使用生产数据和通过特定条件生成新数据这两种方法的一个结合。这种方法提供了由控制因素进行调整的测试数据。

以下是某个大型互助基金在实现测试自动化时采用的策略。如果没有可预测的可重复数据，跨版本的回归自动化测试是没有任何意义的。尽管可以通过联机接口创建大部分的数据，比如资金、客户、账户等，但是还是要有一些数据只能从生产中提取。举例来说，测试账单和税款报告都需要历史交易的记录，除了通过多重的执行周期外没有别的方法可以创建它们。因此，有选择

性地从生产中提取数据并且对测试进行必要的维护这种方法经验证是比较省时的。一旦数据被汇集上来就可以复用。

这同样不是一个容易的过程。你必须克服内聚性的挑战，保证获得的数据子集是有意义的，并且必须具备根据测试条件的需要生成附加数据的有效方法。此外，你必须将测试结果数据作为有价值的财富来对待，建立能够安全保存这些数据的流程，以便以后能够恢复并复用。

尽管复用这个概念已经很普遍且有它的好处，但它同时也带来了一些问题。对于大部分时间敏感的应用程序，不断地复用同样的数据经常是不可行的，除非你能够将数据的日期向前调或者将系统日期往回调。举例来说，一个64岁的员工一个月之后可能就是65岁了，这将导致养老金税率等的变化。

此外，尽管对文件结构和记录布局的任何变更都需要数据转换，但是这个可以看做是一个好的方面，因为我们希望首先在测试环境中对数据的转换进行测试，然后再应用到生产中。

34.4.4 根据数据库生成数据

有了足够的信息来模拟真实的环境时，很明显我们可以用生成的数据建立数据库来进行容量测试和性能测试。如果需要保证数据库的设计能够支持数百万的用户或者数十亿的业务并保证响应时间在可接受的范围内，根据数据库生成数据可能是唯一有意义的一种做法了。

测试数据生成器首先要创建文件或者数据库的描述。大部分情况下，工具可以直接从数据库表中读出相应的字段以及它们的类型、长度和格式。接下来用户可以添加控制有效数据生成的规则、关系以及约束条件。

同时要提供一些标准的数据模板，可以自动生成数十亿的名称、地址、城市、州、邮政编码、社会保障编码、测试日期和其他的一些常用数据，比如随机数、范围和混合类型。大部分产品中还有用户定制的数据类型，可以用来生成唯一的SIC（standard industrialization classification，工业标准分类）业务编码、电子邮件地址和其他数据类型。

一个更重要的也是更难实现的特征就是在复杂的数据库中实现对父子等关系的支持。举例来说，一个父记录，比如一个用户的主账户，必须和很多的子记录相关联，比如不同的分账户和一些交易。这种类型的功能是对参照完整性非常重视的关系数据库环境的基础。

一些用户发现用测试数据生成器生成数据，然后自动化测试工具把这些数据输入到应用程序中，是非常简单的事情。这是一个数据生成和播种的有趣结合。测试数据生成和测试自动化工具之间的结合是很自然的，在一些情况下测试数据的生成是集成在测试执行产品中的。

数据库不仅仅包含数据，还有存储过程和其他表或数据库相关联的外键。在这种情况下，直接生成数据并向数据库表中直接添加是不可行的。经常对数据库的完整性进行维护本身就是一个项目。

当然，容量对数据库本身是一个挑战。更多并不一定就是好事。过多的数据需要很长的创建时间，需要很多的存储空间，并且可能比数据不足引起的问题还要多。

34.4.5 一种基于需求的有效测试用例生成器

因为大部分测试工作需要数百（即使不是数千）个测试用例，所以在捕获/回放工具使用脚

本语言时,大量的开发工作就是这样的结果。在对1个小时的手工测试进行自动化可能需要10个小时的编码和调试时,这就是时间的浪费。这其实是在测试周期中增加了另一份开发工作,这部分工作是计划外、没有人员、没有预算的。对于没有编程背景的测试人员来说,学会如何使用这些工具需要经历一个比较艰难的学习过程。

在无法避免对应用程序的修改时,哪怕是很小的修改,也可能对自动化测试库产生很大的影响。修改一个广泛使用的功能,可能会影响几十个测试用例,甚至更多。修改不但必须映射到受影响的脚本,以及已做的任何必要的修改,而且对结果也必须进行测试。最终,维护工作量会占用测试周期时间的很大一部分,以至于为了满足最后期限的要求,测试人员被迫回到手工测试。这种情况下,工具被束之高阁。

未来自动化测试工具的焦点是具有业务视角而不是编程视角。业务分析可能会从业务的视角来使用这种工具对应用程序进行测试,而不必写测试脚本。

不是要求学会脚本语言,或是记录其测试,以便有人可以将其编写为脚本,大多数高级自动化测试工具可以让记录和自动化一步完成,无需编程工作。对业务有一定了解的应用专家可以学会通过下拉式菜单来开发和执行健壮的自动化测试。

正如前面指出的那样,测试自动化的焦点已经放在了为了工作获得脚本上。但是数据从何而来呢?测试人员如何知道脚本使用的是正确的测试数据呢?测试人员如何知道这些数据覆盖了足够的需求呢?

404

这是到目前为止捕获/回放自动工具缺少的基本要素。没有考虑的是给脚本的测试数据输入和测试场景的质量。通常自动化测试工具脚本程序被分配给测试部门。因为这是非常特别的、精深的编程工作,焦点是获得能正确工作的脚本。假设有人会提供测试数据/测试场景。

下面描述一种直接从测试对象和需求导出测试数据的工具。这类工具的输出是自动化测试工具的输入。这样一种工具跨越了需求阶段和测试阶段之间的鸿沟。

Smartware科技公司的SmartTest是这种未来工具的一个示例,具有下列主要功能。

- 可以使用参数和值的手工输入或从Excel导入。
- 产生成对测试数据,作为变量数据的输入输入到下列自动工具(或手工测试)中:
 - 惠普公司的 Quick Test Pro;
 - IBM 公司计算机协会的 CA Verify;
 - Compuware 的 TestPartner;
 - Empririx 公司的 e-Test Suite;
 - Segue 公司的 SilkTest。
- 将需求/业务规则动态地应用到成对测试数据。
- 提供测试用例和需求(业务规则)之间的可追溯性。

图34-1给出了SmartTest树结构,测试用例用这种树结构组织和安放。图中还给出了类似Excel表的格式,表中包含参数及相关联的值。在这个例子中,参数是GUI应用中的行,和各自的值放在一起。这些参数可以手工输入也可以通过目标应用GUI的屏幕来输入。

Test Data				Requirements		Results
AGE	EMPLOYMENTSTATUS	SELFEMPLOYMENTDURATION	EMPLOYMENTDURATION	PRIMARYAPPLICANTCREDITSCORE	JONTAPPLICANT	PROPERTYTYPE
20	Employed	Less than 2 yrs	Less than 6 mo	90000	Yes	Single Family Home
21	Self-Employed	2-4 yrs	6 mo - 1 yr	100000	No	Town House
22	Unemployed	Greater than 4 yrs	greater than 1 yr	250000		Condo
65				750000		Duplex
				1000000		Commercial
				2000000		

图34-1 测试用例树、测试参数和值

图34-2给出了SmartTest业务规则构建器。业务规则可以通过下面的方式输入：点击预定义参数及相关联的值来构建结构化英语if-then-else格式。内置的参数被称为“期望结果”，可以用规则来自定义从测试获得的期望结果。

Test Data		Requirements		Results
Rule ID	Include	Rule Type	Condition	
Rule1	Yes	Require	If the 'AGE' is less than 21 Or ('EMPLOYMENTSTATUS' is equal to Employed And 'AGE' is greater than 65)	
Rule2	Yes	Require	If the 'EMPLOYMENTSTATUS' = Self-Employed And 'SELFEMPLOYMENTDURATION' = Less than 2 yrs	
Rule3	Yes	Require	Whenever 'EMPLOYMENTSTATUS' is equal to Employed And 'EMPLOYMENTDURATION' is equal to 6 mo	
Rule4	Yes	Require	When 'EMPLOYMENTSTATUS' is equal to Unemployed	
Rule5	Yes	Require	Whenever the 'PRIMARYAPPLICANTCREDITSCORE' is less than or equal to 550 Or 'PRIMARYAPPLICANTCREDITSCORE' = None	
Rule6	Yes	Require	Whenever the 'PROPERTYTYPE' is equal to Commercial	
Rule7	Yes	Require	When the 'APPRAISEDVALUE' is greater than 1000000	
Rule8	Yes	Require	Whenever the 'APPRAISEDVALUE' is less than 100000	
Rule9	Yes	Require	When the 'LOANAMOUNTREQUESTED' is greater than (.85 * 'PURCHASEPRICE')	
Rule10	Yes	Require	If the 'LOANAMOUNTREQUESTED' is less than 300000 And 'MORTGAGETYPE' = 30 yr Jumbo Or 'MORTGAGETYPE' = 5/1 Jumbo ARM	
Rule11	No	Require	When 'JONTAPPLICANT' is equal to No And 'EXPECTED RESULT' <> Failed	
Rule12	Yes	Require	If 'JONTAPPLICANT' is equal to Yes And 'EXPECTED RESULT' is not equal to Failed	
Rule13	No	Exclude	When 'EMPLOYMENTSTATUS' = Employed And ('SELFEMPLOYMENTDURATION' = Less than 2 yrs or 'SELFEMPLOYMENTDURATION' = 2-4 yrs or 'SELFEMPLOYMENTDURATION' = 4 yrs)	
Rule14	No	Exclude	When 'JONTAPPLICANT' is equal to No And 'JONTAPPLICANTINCOME' > 8	

图34-2 结构化英语业务规则

图34-3给出了SmartTest如何以参数和值的成对交互来生成测试用例和测试数据。每一行是一个测试用例，业务规则动态地应用于每一个测试行。这调整了测试，以反映需求（或业务规则），正负测试数据都可以被构建。

Test Data			Requirements		
TestNo	AGE	EMPLOYMENTSTATUS	SELFEMPLOYMENTDURATION	EMPLOYMENTDURATION	PRIMARYAPPLICANTINCOME
Test1	20	Unemployed	Greater than 4 yrs	greater than 1 yr	250000
Test2	20	Self-Employed	2-4 yrs	6 mo - 1 yr	1000000
Test3	21	Self-Employed	Greater than 4 yrs	6 mo - 1 yr	1000000
Test4	21	Self-Employed	Greater than 4 yrs	Less than 6 mo	100000
Test5	22	Self-Employed	Greater than 4 yrs	Less than 6 mo	90000
Test6	22	Unemployed	2-4 yrs	Less than 6 mo	250000
Test7	22	Self-Employed	Less than 2 yrs	greater than 1 yr	1000000
Test8	22	Unemployed	2-4 yrs	greater than 1 yr	2000000
Test9	66	Self-Employed	2-4 yrs	Less than 6 mo	750000
Test10	66	Unemployed	Greater than 4 yrs	6 mo - 1 yr	1000000
Test11	66	Unemployed	2-4 yrs	greater than 1 yr	90000
Test12	22	Unemployed	Less than 2 yrs	6 mo - 1 yr	90000
Test13	20	Unemployed	2-4 yrs	Less than 6 mo	250000
Test14	21	Employed	Greater than 4 yrs	6 mo - 1 yr	750000
Test15	66	Employed	Greater than 4 yrs	greater than 1 yr	100000
Test16	20	Self-Employed	Less than 2 yrs	Less than 6 mo	250000
Test17	21	Unemployed	Less than 2 yrs	6 mo - 1 yr	750000
Test18	22	Unemployed	2-4 yrs	greater than 1 yr	1000000
Test19	66	Employed	Greater than 4 yrs	Less than 6 mo	2000000
Test20	20	Self-Employed	2-4 yrs	6 mo - 1 yr	2000000
Test21	21	Unemployed	Greater than 4 yrs	greater than 1 yr	90000
Test22	20	Unemployed	2-4 yrs	Less than 6 mo	90000
Test23	22	Self-Employed	2-4 yrs	greater than 1 yr	100000
Test24	66	Unemployed	Greater than 4 yrs	greater than 1 yr	250000
Test25	66	Self-Employed	2-4 yrs	6 mo - 1 yr	750000
Test26	21	Self-Employed	2-4 yrs	6 mo - 1 yr	100000
Test27	22	Unemployed	Greater than 4 yrs	greater than 1 yr	250000
Test28	22	Unemployed	Greater than 4 yrs	greater than 1 yr	250000
Test29	20	Unemployed	Greater than 4 yrs	greater than 1 yr	2000000
Test30	22	Unemployed	Greater than 4 yrs	greater than 1 yr	250000
Test31	22	Unemployed	Greater than 4 yrs	greater than 1 yr	1000000
Test32	20	Self-Employed	2-4 yrs	6 mo - 1 yr	1000000
Test33	22	Unemployed	Greater than 4 yrs	greater than 1 yr	250000
Test34	66	Unemployed	Greater than 4 yrs	greater than 1 yr	750000
Test35	22	Self-Employed	2-4 yrs	6 mo - 1 yr	2000000

Business Rules
☒ All
☐ Selected
☐ None

☒ Optimize

Show Test Case
☐ Positive
☐ Negative
☒ All

TEST RESULTS:
Total test combinations = 10883911680
Number of optimized tests = 37

Export To Excel
Print

图34-3 逐对产生的测试数据

图34-4给出了双向SmartTest报告，这个报告与哪一个测试用例与每条业务规则相关联有关，这一特性在维护期间对识别根据需求的变化应该被执行的测试用例特别有用。（更详细的内容请参见www.smartwaretechnologies.com。）

Test Data				Requirements								Result
	BUSINESS		RULES									
TESTS	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
1								x				
2												
3	x			x								
4												
5	x					x				x		
6				x			x			x		
7								x				
8												
9								x		x		
10				x								
11												x
12				x								
13												x
14		x				x	x					
15							x		x			
16						x			x			
17				x		x		x				
18				x						x		
19										x		
20		x										
21				x								
22									x	x		
23							x		x			
24				x								
25						x		x				
26		x										
27				x								
28										x		

图34-4 可追溯性矩阵（业务规则与测试用例）（参见www.smartwaretechnologies.com）

35.1 测试工具选择检查表

选择适合的工具是件很难的工作。在选择工具前首先要回答几个问题。F.19节列出了一些有助于质量保证小组评估和选择自动化测试工具的问题。

下面根据工具自身的目标和特性，对目前所使用的测试工具种类进行了分类。在35.2节中，对常见的厂商所支持的每种工具进行了讨论。

- 功能/回归测试工具。这类工具通过原始的图形化用户界面帮助你对软件进行测试，以确保系统的功能。
- Bug管理工具。这类工具有助于跟踪软件产品缺陷和管理产品增强请求，管理缺陷状态，从缺陷发现到结束。
- 测试过程/管理工具。这类工具有助于在命令行、API或者协议级别组织和执行测试用例集。某些工具拥有图形用户界面，但对测试具有原始图形用户界面的产品没有特殊的支持。
- 需求分析工具。这类工具有助于验证完整性，并且定位有歧义和冲突的需求。
- 单元测试工具。这类测试有助于对软件进行单元测试，这通常由开发人员通过被测软件公共界面下的接口来完成。
- 负载/性能测试工具。这类工具有助于分析在不同负载和压力下进行测试时系统的性能。
- 测试数据生成工具。这类工具有助于创建测试数据和测试用例。
- 现场监控工具。这类工具有助于在整个IT服务提交生命周期中测量和最大化值，以确保应用满足质量、性能和可用性目标。
- Java测试工具。这类工具有助于测试Java网站小应用程序。
- 嵌入式测试工具。这类工具有助于验证在底层设备上运行的系统，比如视频芯片。
- 数据库测试工具。这类工具有助于验证数据库完整性、业务规则、访问和刷新能力。
- Web测试工具。这类工具有助于定位中断的网络链接，评价在大负载的情况下基于Web的系统的性能。
- 安全测试工具。这类工具有助于评价系统确保系统完整性和保护资源的能力。

35.2 商业厂商工具描述

表35-1提供了一些商用测试工具的概况，介绍依照字母序排列。各工具的名称被列出并被交叉引用到支持的软件测试类型中。

表35-1 测试工具名称与工具种类^a

工具名称	功能 回归测试工具	bug 管理工具	测试过程 管理工具	需求分析工具	单元测试工具	负载 性能测试	测试数据生成工具	现场监控工具	Java 测试工具	嵌入式测试工具	数据库测试工具	Web 测试工具	安全测试工具
Abbot									×				
Aberro Test	×												
AccVerify/AccRepair												×	
actiWATE	×												
AdminiTrack		×											
ADT Web		×											
AETG Web	×						×						
AgileTest									×				
Application Center Test						×							
Application Manager								×					
AppPerfectSevSuite									×				
ApTest Manager			×										
Aqitator									×				
Atomic Watch								×					
AutoIT	×												
Automated Test Designer	×												
AutomatedQA	×												
AutomateTestManager	×												
Bug/Defect Tracking Expert		×											
BugAware		×											
Bugcentral		×											
BugHost		×											
BugHuntress	×												
Bugkilla									×				
BugRoster		×											
BugStation		×											
BugTimer						×							

(续)

工具名称	功能 回归测试工具	bug 管理工具	测试过程 管理工具	需求分析工具	单元测试工具	负载 性能测试	测试数据生成工具	现场监控工具	Java 测试工具	嵌入式测试工具	数据库测试工具	Web 测试工具	安全测试工具
GJTester									×				
GUIDancer									×				
GUITAR	×												
Haven	×												
HTML Candy												×	
HTML PowerTools												×	
HTML Tidy												×	
httpUnit	×												
Infocus												×	
IssueTrak		×											
Jcover									×				
Jenny											×		
JIRA		×											
JStyle									×				
JSystem									×				
Jtest									×				
JTrac		×											
Jumpstart											×		
JUnit									×				
JVerify									×				
KCC									×				
KenTestMan			×										
Link Checker Pro												×	
Link Runner												×	
Link Validator												×	
LinkScan												×	
LinkSleuth												×	
LISA									×				
Loadea Test						×							
LoadRunner						×							
Lorem Ipsum Generator							×						
Mantis		×											

(续)

[illegible]

(续)

工具名称	功能 回归测试工具	bug 管理工具	测试过程 管理工具	需求分析工具	单元测试工具	负载 性能测试	测试数据生成工具	现场监控工具	Java 测试工具	嵌入式测试工具	数据库测试工具	Web 测试工具	安全测试工具
Rational Robot					×								
Rational Test Manager			×										
Reactis Tester										×			
Real Validator												×	
RegressionTester	×												
Replay Xcessory	×												
Repro	×												
RR Tracker		×											
SAP Software Quality Assurance Testing Tool	×												
ScriptMap	×												
ScriptTech	×												
SeeDEV		×											
Shunra\Storm						×							
SilkCenter Test Manager			×										
SilkPerformer						×							
SilkTest	×												
SilkTest	×												
SiteMonitor								×					
SM2SMV				×									
Smalltalk Test Mentor	×												
*SMARTCHECK				×									
*SMARTPROCESS			×										
*SMARTTEST	×						×						
SPRAT				×									
SQL DB Validator											×		
SQL Profiler											×		
SQL/Test Professional											×		
Squish		×											
TALC2000	×												
Task Complete		×											
TBrun										×			

(续)

工具名称	功能 回归测试工具	bug 管理工具	测试过程 管理工具	需求分析工具	单元测试工具	负载 性能测试	测试数据生成工具	现场监控工具	Java 测试工具	嵌入式测试工具	数据库测试工具	Web 测试工具	安全测试工具
WinFeedback													
WinRunner®	×												
Woodpecker IT		×											
XtremeLoad													
X-Unity					×								
yKAP		×											

35.3 开源自由件厂商工具

表35-2提供了某些开源软件测试工具的概况，介绍依照字母序排列。各工具的名称被列出并被交叉引用到支持的软件测试类型中。

表35-2 开源测试工具与工具类型

工具名称	功能 回归测试工具	测试设计 数据工具	负载 性能测试工具	测试过程 管理工具	单元测试工具	缺陷管理工具	安全测试工具
Abbot Java GUI Test Framework	×						
ALLPAIRS	×						
Anteater	×						
Anthill Bug Manager						×	
Apache JMeter			×				
Apache Postage	×						
Apodora	×						
Appache Jelly					×		
Arbiter	×						
AUnit					×		
Autonet	×						
Avignon	×						

(续)

工具名称	功能 回归测试工具	测试设计 数据工具	负载 性能测试工具	测试过程 管理工具	单元测试工具	缺陷管理工具	安全测试工具
benerator			×				
BFBTester							×
BTsys						×	
BugBye						×	
Buggit						×	
Bugs Online						×	
Bugtrack						×	
Bugzilla						×	
Canoo WebTest	×						
CLIF			×				
Codestriker					×		
CROSS							×
Crosscheck	×						
csvdifff	×						
curl-loader			×				
Data Generator		×					
DBFeeder	×						
DbFit	×						
DBMonster			×				
DbUnit					×		
DejaGnu	×						
Deluge			×				
Dieseltest			×				
dogtail	×						
Doit	×						
DTraq					×		
EasyMock		×					
Eclipse TPTP	×						
EMOS Framework	×						
Enterprise Web Test	×						
Ethereal							×
Expect	×						

(续)

工具名称	功能 回归测试工具	测试设计 数据工具	负载 性能测试工具	测试过程 管理工具	单元测试工具	缺陷管理工具	安全测试工具
Faban			×				
FindBugs					×		
Firefox Web Developer							×
Fitness				×			
Flawfinder							×
Frankenstein	×						
FunkLoad			×				
GITAK	×						
GNU/Linux Desktop Testing Project	×						
Grinder			×				
Hammerhead 2			×				
Harness	×						
http_load			×				
Httpperf			×				
IBM®Optim		×					
IdMUnit	×						
Imprimatur	×						
IMS						×	
ItiN	×						
ITP	×						
ItsNat	×						
iValidator	×						
Jacobie	×						
Jameleon	×						
JChav			×				
JCrawler			×				
jDiffChaser	×						
Jemmy	×						
JFunc	×						
JMockit		×					
JWebUnit	×						
Latka	×						

(续)

工具名称	功能 回归测试工具	测试设计 数据工具	负载 性能测试工具	测试过程 管理工具	单元测试工具	缺陷管理工具	安全测试工具
Linux Test Project	×						
Linux Test Project test suite			×				
Lobo			×				
LogiTest	×						
LReport	×						
MActor	×						
Mantis						×	
MaxQ	×						
Metasploit							×
MozUnit	×						
Nessus							×
Nikto							×
NTime			×				
Oliver	×						
OpenSTA			×				
OpenWebLoad			×				
org.tigris.mbt	×						
Pamie	×						
Paros							×
Pounder	×						
ProofPower					×		
P-unit							
Pywinauto			×				
qaManager	×						
QAT				×			
QaTraq	×						
QMTest				×			
Roundup	×						
rth						×	
RTH Turbo				×			
Sahi				×			
Samie	×						

(续)

工具名称	功能 回归测试工具	测试设计 数据工具	负载 性能测试工具	测试过程 管理工具	单元测试工具	缺陷管理工具	安全测试工具
Seagull	×						
Selenium			×				
SharpRobo	×						
Siege	×						
Sipp			×				
Slamd			×				
Soap-Stone			×				
soapUI			×				
Solex	×						
STAF	×						
stress_driver	×						
Systin			×				
tclwebtest	×						
TCW	×						
Tesly				×			
Test Environment Toolkit				×			
TestGen4Web				×			
Testitool	×						
TestLink				×			
TestMaker				×			
Testmaster			×				
Testopia				×			
TestTest				×			
tg	×						
Toaster					×		
Tomato	×						
TPTEST	×						
Tsung			×				
Valgrind			×				
Visual Parser			×				
Watij					×		
WatiN	×						

(续)

工具名称	功能 回归测试工具	测试设计 数据工具	负载 性能测试工具	测试过程 管理工具	单元测试工具	缺陷管理工具	安全测试工具
Watir	×						
Web Application Load Simulator	×						
Web Form Fuzzer			×				
Web Polygraph	×						
WebGoat			×				
WebInject							×
WebLOAD	×						
WebScarab			×				
WebTst							×
WET	×						
Win32-IEAutomation-0.5	×						
Wireshark	×						
XmlTestSuite	×						×

35.4 应该考虑使用测试自动化的情况

测试工具的使用取决于测试目标。只有当手工测试无法完成的时候才会研究使用适合的测试工具，这是使用工具的一般原则。例如，对系统进行压力测试时，需要一组测试人员同时登录系统以模拟系统的最大负荷量。但是这种方法存在局限性，即不能精确或重复地测量并发性能。对于这种情况，使用负载测试工具就可以在有限压力的条件下模拟多个虚拟用户。

在下面这些情况下可能会需要回归测试工具。

- 对应用程序的每个版本都要进行测试的情况。这种情况下人力资源的使用比较耗时、不可靠或不一致。
- 测试时同一操作需要使用多个数据值。
- 测试时需要如SQL和GUI属性等系统内部详细信息。
- 需要对系统进行施压以检测其性能。

使用测试工具有下列益处。

- 比手工测试速度快。
- 运行时不会受到人为干扰。
- 在测试运行后提供代码覆盖分析。

- 可精确地反复重现。
- 可复用，如编程中使用的子程序。
- 详细测试用例（包括可预见的“期望结果”）来源于功能规约或技术设计文档。
- 带有数据库的稳定的测试环境可以恢复到初始状态。这样，在每次修改应用程序的时候都可以重复使用测试用例。

35.5 不应该考虑测试自动化的情况

无论是由于业务原因还是由于自动化工具和自动化项目上需要投入的大量资金、时间和精力，目前大多数的测试还是会采取手工方式进行。为什么？有三个原因导致测试自动化并不是很成功：学习曲线不合理、需要投入的开发工作量以及日常的维护支出。

关于学习曲线问题最简单：虽然传统的测试脚本工具基本上使用一些特定的编程语言，但是即使是最好的测试工程师也不过是一个应用专家而不是一名程序员。

这就产生了技术上的代沟，而跨越这道代沟则需要完成不合理的学习曲线。因为业务知识成为优秀的测试人员的应用专家不太可能具有编程技能。获得这些技能可能会需要一年半载，而且如果不具备这些技能将会降低脚本库的可维护性。

大部分测试工具的供应商都明白工具的这个弱点，他们也试图通过捕获/回放功能来解决这个问题。表面上看来，测试人员似乎可以把手工测试自动“录制”成一个可以回放的测试脚本，这种方法也似乎解决了学习曲线的问题，但是事实上，它造成的问题常常比它所解决的问题更严重。

428

首先，录制好的测试脚本不够完善，很容易导致失败。由于没有错误处理或者逻辑处理，所以哪怕是应用程序行为或数据的最小偏差也会导致脚本异常中止或错误。此外，它将脚本和数据结合到单个程序中，所以没有了可复用性和模块性。本质上来说，其最终结果就是非结构化、设计质量差的代码。

同样地，对于修改和维护来说也不像录制脚本那样简单。由于软件的原因改变了某些地方，其相应的脚本也必须进行修改。进行大量的脚本修改及错误调试是件耗时而复杂的工作。

一旦公司发现使用捕获/回放不是长久之计，那么他们就得要么放弃使用要么投入开发力量了。

相对于对工具的广泛信任而言，有些时候购买测试工具并不是明智之举。限制测试工具使用的因素有以下几个。

- 不切实际的期望——IT产业总是会把一项新技术解决方案当成是万能药。人们习惯于对新技术持有乐观态度，销售人员会把他们的工具说得天花乱坠。其结果就是导致不切实际的期望。
- 缺少测试规程——测试自动化的前提条件是要存在明确的手工测试过程。缺少好的测试习惯及标准对于测试自动化有百害而无一利。只有在适当的时候制定出完美的测试计划，自动化测试工具才能自动地查找缺陷。
- 安全误区——即使一组测试脚本的成功运行也不能保证自动化测试工具检测出了所有缺陷。因为相反的假设是会导致安全误区的，自动化测试的好坏和你的测试用例和测试输入数据息息相关。
- 技术难点——自动化测试工具有其自身的缺陷。技术环境的变更如操作系统的变化会严

重限制自动化测试工具的使用。

- 组织上的问题——测试自动化还会受到组织的影响，因为自动化测试可能需要跨项目甚至跨部门。例如，数据驱动测试脚本的使用通常需要测试时输入Excel表形式的行数据。这些数据可能是由其他小组提供的，比如业务系统分析部门，而不是测试部门。
- 成本——组织可能无法承担测试工具的费用，比如成本和性能之间的权衡。
- 文化——由于需要适合的技能 and 长期的质量义务，开发文化可能还没有做好接受测试工具的准备。
- 易用性测试——没有自动化测试工具可以测试软件的易用性。
- 一次性测试——只执行一次的测试没有必要花费时间和精力去使用测试工具。
- 时间紧迫——如果时间紧任务重，使用测试工具可能不是明智之举。因为使用测试工具首先要花费时间去学习、安装并将其结合到开发中。
- 特别测试——没有正式测试设计及测试用例的回归测试工具是无用的。
- 预期结果——如果不能提供可预期的结果，就没有必要使用回归测试工具。
- 不稳定性——如果每一个测试周期内，系统都有一些巨大改动，那么花费过多的时间去维护回归测试是得不偿失的。

429

430

本章会简要地描述在获取、实现和使用测试工具过程中的一些步骤。管理大型项目要求所有工作都必须分解成任务，这样可以定义完整的标准。因为任务之间的转移按照步骤进行，为了使这些活动按顺序执行，需要预先确定这些步骤的进度。下文所描述的步骤为这样的进度提供了一个通用的大纲。实际的进度取决于许多因素，这些因素需要根据每个具体工具的使用来决定。

36.1 步骤 1：定义测试需求

要实现的目标应该以某种格式描述，以便以后确定这些目标是否得到实现（即步骤15）。典型的目标包括把C++程序的平均处理时间降低五分之一，实现程序和数据集与其他组织完全的交换性，按照现有标准调节文档格式等。目标陈述也应该标明职责，特别是高层人员的职责，并详细说明与其他组织的协调需求。如果采用了集中管理方式，目标陈述还可能包括预算和预计的完成日期。一旦明确了这些限制，财务管理部门才能够批准采购计划并委派下面的人去落实。

431

36.2 步骤 2：设定工具目标

步骤1所生成的目标应该翻译为指定开发和操作环境下对工具的功能期望和需求。在这一步中还需要添加对工具成本和可用性的限制。例如，对一个程序格式控制工具的典型期望是提供头部标识、统一缩进、协助打印列表以及对所有Pascal程序分别注释等功能。此外，工具必须在购买组织的计算机和操作系统上运行过。尽量考虑那些已经商用1年以上并且在N种场合都曾经使用过的工具。（N的值由组织所拥有的场合数量来预先确定。）

36.3 步骤 3a：非正式采购模式下的选择活动

如果计划采用非正式的采购模式，应该采用以下步骤。

36.3.1 任务 1：制定采购计划

采购计划是沟通软件管理者上下级之间行动的命令链条。计划也可以包括工具目标的描述（步骤2）。采购计划包括工具引入阶段的预算和后续步骤的安排，证明预期收益与所需资源的关系，预计其他组织对引入工作的帮助（例如，工具本身、修改补丁或者培训资料等），组织内部各个后续事件责任的分配，特别要标明使用软件的工程师。在计划中还需要描述基本工具文档需求。

36.3.2 任务 2：定义选择标准

选择标准需要列出一些能够提高工具使用效率的属性。典型的选择标准包括以下几项：

- ☐ 能够实现特定工具目标；
- ☐ 易于使用；
- ☐ 易于安装；
- ☐ 处理时间最短；
- ☐ 与其他工具兼容；
- ☐ 价位低或成本便宜。

432

虽然需要更深层次地考虑这些标准以进行目标评估，但是这个步骤可以留给那些评分的人去考虑。先前已经提出或者该步骤中生成的限制应该和标准一起进行汇总。

36.3.3 任务 3：确定候选工具

这是软件工程师负责处理的第一个步骤。确定候选工具列表首先要有全面的工具目录。通常需要准备两个列表，第一个包括所有满足功能要求的工具而不必考虑限制。对于可能的候选工具，开发人员需要给出书面意见，并检查是否符合给出的限制。此时，就会生成第二个列表，其中包含既满足功能性需求又符合限制的工具。如果这个列表太短，我们就可以适当放宽一些限制。

36.3.4 任务 4：进行候选工具评审

用户必须评审由软件工程师准备的候选工具列表。因为用户往往对软件工具不了解，软件管理人员会提出以下一些特殊问题。

- ☐ 该工具能否处理当前的文件格式？
- ☐ 工具命令和编辑器命令是否一致？
- ☐ 需要多少培训？

评审通常需要很长的时间，所以需要指定一个明确的截止日期。因为用户常常认为这是一项优先级比较低而且长期的任务，需要大量的后续跟进管理。如果可能，最好获得工具的试用版本，或者能够安排在其他环境下使用演示版。

36.3.5 任务 5：为候选工具打分

对于任务2中标明的每一条标准，应该根据厂商说明中的基本信息、工具演示、用户评审、工作环境中的观测以及以前用户的意见这些因素给每种工具打分。一旦确定了每个标准的加权因子，就可以得到代表每种工具产品的总得分。一旦设定了标准的优先级，那么评分工作就剩下根据每项标准的对工具进行评分了。这样工具的优劣就一目了然了。

433

36.3.6 任务 6：选择工具

做决定的任务是预留给软件经理的，他们可以评审得分并考虑标准范围之外的一些因素。例

如，某个代理商的报告说选中的供应商不能提供完善的服务。如果确实选中的工具得分不高，那么软件工程师就得全面评审这个工具的所有功能以避免意外的安装问题。（非正式采购模式至此为止，剩余的过程从步骤4开始继续。）

36.4 步骤 3b: 正式采购模式下的选择活动

如果正式的工具采购计划有效，应该执行下列任务。

36.4.1 任务 1: 制定采购计划

计划必须包括步骤3a的任务1中提到的所有元素，还应该包括采购过程的限制以及所有采购文档的详细职责（例如，工作说明和提案所需的技术、管理规定等）。

36.4.2 任务 2: 创建技术需求文档

技术需求文档是工具需求和工具必须满足的限制的非正式描述。虽然可以从采购计划中获得大量可用材料，但是还应该给工具用户提供更详细的信息，使得评审能够更有意义。

36.4.3 任务 3: 评审需求

用户必须评审所建议采购的技术需求。正如步骤3a的任务4中提到的，可能要提醒用户一些相关问题，并且要有密切的跟进管理以提供及时的响应。

36.4.4 任务 4: 生成请求建议

技术部分的请求建议应该来自于技术需求文档并且用户可以添加注释。技术上的注意事项一般包括下列几项。

- 工具正式发布的规约，包括适当的文档、操作环境的限制以及质量保证条款等。
- 所购工具的工作说明。这包括运行工具所需的标准过程（如工具的配置管理）以及所要安装的工具的文档或测试报告。培训和操作的支持需求也在工作说明中确定。
- 所建议的评估标准及格式需求。将这些标准列出来是很重要的事情。也可以确定这些标准的子因素，并将所建议格式的所有限制条件都包含在内（例如，主标题、页数或者期望样本输出值等）。

36.4.5 任务 5: 简化建议

该活动应该由管理人员来执行。大多数采购组织都维护了潜在资源的列表。如果软件组织知道了潜在的投标候选人，就应当将这些投标者的名字告诉采购部门。采购部门应该审阅这些反馈，以符合提案所需的主要法律规定。

36.4.6 任务 6: 进行技术评估

应按照预先确立的标准对接收到的每个提案进行评估。没有达到主要技术需求提案就会彻底

失效。给这些有可能达到标准的建议打分，这一分数将与管理人员单独评估的成本和进度因素放在一起加以考虑。

自动化测试工具可能需要针对测试自动化环境定制。为了演示测试工具的能力以及与应用程序的兼容性，应该从供应商获得POC。在POC中，业务环境应该利用工具进行自动化，覆盖各种业务验证点和动作。

36.4.7 任务 7：选择工具来源

以成本、进度及技术等要素的组合为基础选择工具来源。如果不是最高等级的技术提案，那么经理必须要求软件经理和软件工程师进行额外的评审来决定是否采用这个工具了。（来源选择是正式采购模式下的选择活动的最后一步，剩余的过程从步骤4开始。）

435

36.5 步骤 4：采购测试工具

为了确保所选工具的费用符合所批的预算，采购过程需考虑使用许可是否充足、合同约定、相关采购法规的限制等。供应商必须提供源程序，满足特殊测试和性能需求，并维护工具。在非正式的采购中，可以考虑试用一段时间。

如果采购计划表明需要进行外部培训，则需对供应商提供培训的能力和联合购买工具及培训的成本优势进行调查。如果同时采购工具和培训可以节省资金，则可以在外部培训需求确定后进行采购（步骤7）。

36.6 步骤 5：制定评价计划

评价计划基于步骤1所确定的需求目标和步骤2的工具目标，描述了为了选择一款特定的软件，怎样评估软件是否达到这些目标。计划中所包含的传统项目为安装的里程碑和日期以及初始操作能力和后续提高的性能级别。当期望生产能力、响应时间或周转时间得以改善时，需要对包含这些数据的报告进行验证。测试职责、报告及其他工作必须在计划中进行分配，评估报告的主题要点也应包括在内。

验收测试规程是评估计划的一部分，尽管对于主要工具的采购来说它可能只是单独的一个文档。这个规程列出了必须依照采购供应对收到的工具进行测试的详细步骤，与计算机环境一起对工具的交互作用进行评价（例如，对吞吐量的不利影响），并生成验收报告。

36.7 步骤 6：制定工具经理的计划

工具经理的计划描述了怎样选择工具经理、对所采纳工具所承担的责任以及所需要的培训。工具经理应该是有经验的系统编程人员，对当前的操作系统比较熟悉。需要以文档审核的形式对选定的工具进行操作和安装的培训，拜访工具的用户或者由供应商提供的培训都应该妥善安排。软件工程师对工具经理的计划负责，工具经理应该在软件工程师的指导下工作。工具经理的计划应该由软件管理层来批准。

436

36.8 步骤 7: 创建培训计划

培训计划首先应该考虑工具本身附带的培训资料,例如文档、测试用例、在线诊断等。供应商提供的标准内部培训可以作为补充,例如音频或者视频教学课程。考虑到成本,正式培训应该在没有其他措施的时候采用。员工接受正规培训应该在计划中明确,以保证场所办公设备准备好,例如充足的电脑终端数量。如果期望工具供应商提供培训,应该尽早提出,以便能够实现培训与工具一起购买(见步骤4)。用户应该介入培训计划的制定,与用户的合作是很有必要的。测试培训计划必须由软件工程师来完成并且获得软件管理部门的批准。如果要用到外部的人员或设备,计划的相应部分也应该提供给采购部门的职员。

36.9 步骤 8: 接收工具

工具由采购部门交付到软件工程师手中。

36.10 步骤 9: 执行验收测试

软件工程师或员工应该在模拟验收的条件下测试该工具,仅做必要的修改,为了让工具能够在主计算机上运行起来。一旦测试报告发布并被软件经理批准,那么这个工具就被正式验收了。

36.11 步骤 10: 召开推介会议

当确定工具已经被满意地验收的时候,软件管理部门应该召集所有使用工具和工具产品(如工具产生的报告或清单)的相关人员参加一个推介会议。直接在会上与大家沟通工具的使用目标(例如增加的吞吐量或清单更加清楚易读等)。评价计划中的重点内容要向大家讲清楚,与引入工具相关联的职责的变化也要在会上描述清楚。要让所有人员认识到引入工具会遇到一些问题,这是允许的。同时,也要提醒大家工具带来的全部益处要很长时间才能被认识到。

437

36.12 步骤 11: 实施修改

这一步骤是由工具经理根据获得批准以后的工具管理计划来完成的,包含对工具和文档的修改,也包含对操作系统得改变。在一些极端的情况下,对计算机的一些修改也是很必要的(如信道的分配)。对工具的一些典型修改包括没用选项的删除,提示信息或诊断内容的改变,还有为在当前环境中高效地使用工具而做的适应性修改。另外,所有修改必须要有完整的记录。

应该仔细审核供应商为工具提供的文档资料,使其适合当前的计算机环境,也要适合对工具所做过的修改。删除一些不再适合的章节,也要增加一些特定的编程环境需要的材料,二者同样重要。没用的一些配置选项也要清晰地标识出来,或者干脆从手册中删掉。如果工具不能应用于一些常见软件(如语言不兼容或与操作系统的接口有冲突),应该在工具的使用手册中插入警告提示信息。

36.13 步骤 12：培训工具的使用户

培训是软件工程师和使用工具的使用户双方共同的责任，做好培训有助于工具使用的推广。软件工程师负责培训的内容（要与批准后的培训计划保持一致），工具使用者控制培训时间长短和日程安排。工具使用户应该能够终止那些没有帮助作用的培训步骤，同时也可以要求对很有帮助意义的部分增加更多解释说明。对工具更多选项或更深功能的培训是必要的，再培训或者培训是与使用者一起探讨使用工具的过程中遇到的一些问题的很好机会。

36.14 步骤 13：在操作环境中使用工具

在操作环境中第一次使用工具的时候，尽量让技术能力比较强的人员参与，并且尽可能少用工具的配置选项。不要把工具的第一次使用放在进度本来就紧张的项目中。使用中出现的问题要在大范围推广使用该工具之前解决。如果第一次使用是成功的，其他大范围的人员就比较容易上手使用，并且对工具的使用也会更加深入。

用户对培训的评论、工具的第一次使用状况以及扩展功能的使用状况都应该提供给软件工程师。需要对用户界面、响应的速度和形式以及计算机资源的利用等进行的改进也是非常合适的话题。在初步使用、使用6个月以及使用1年等几个时间点上都应该有正式的说明。

36.15 步骤 14：撰写评估报告

软件工程师利用步骤5中产生的大致计划来准备评价报告。用户的评论、建议和观察到的实际情况是这份文档的重要组成部分。最重要的是，报告里面必须体现总体的目标和工具的目标是如何实现的。报告还可以包含安装和使用过程中的一些观察资料，还有来自供应商方面安装和培训的合作情况，以及任何学到的经验和教训。

对工具和安装工具的主机的修改也要在报告里体现出来，这些注释性的内容对以后的工具用户很有用。报告必须要获得软件管理部门，尤其是公司管理层的批准。

36.16 步骤 15：确定目标是否实现

管理层收到评价报告以后，应该确定在步骤1中建立的目标是否实现，应该给出一份书面的意见，确定下列方面的问题：

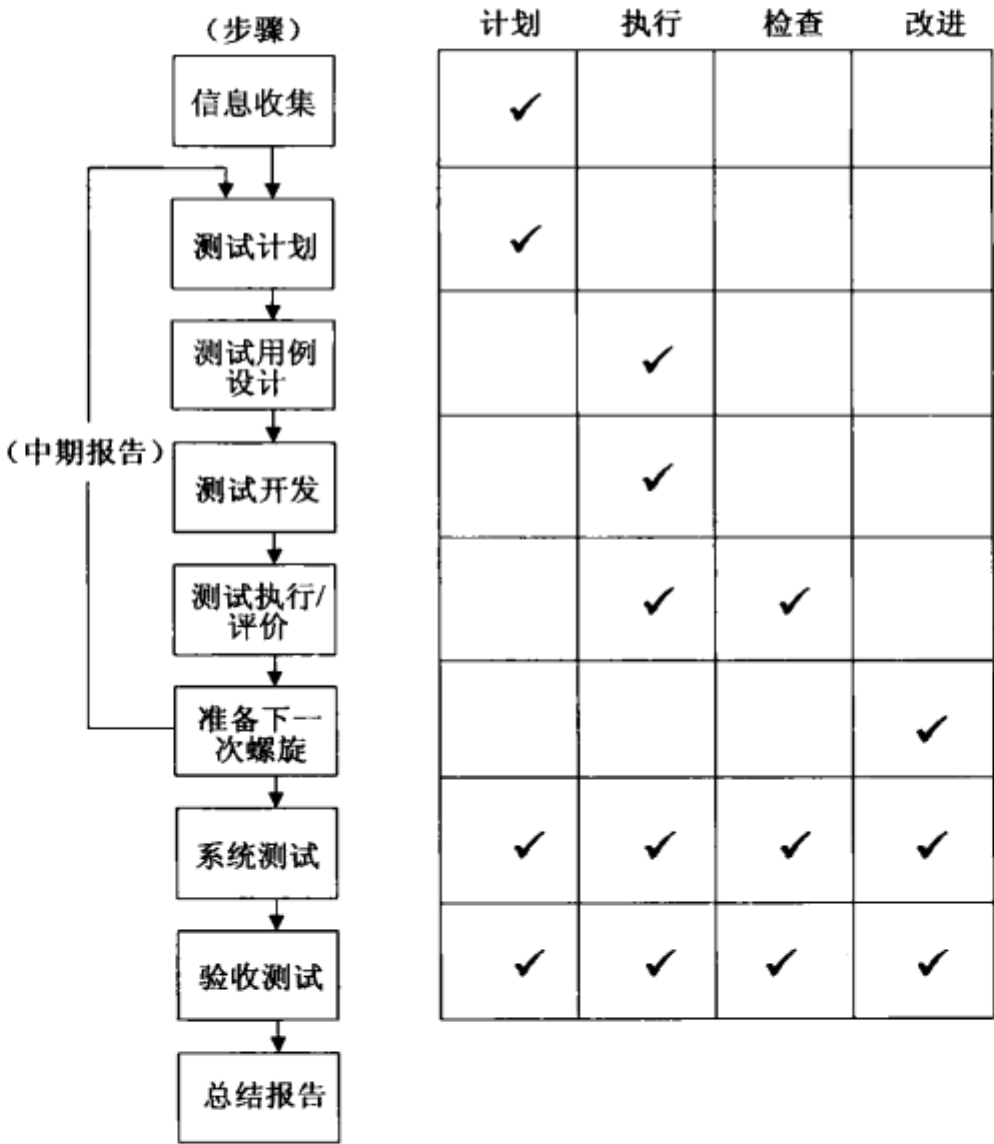
- 技术目标已达到；
- 预算和其他资源方面的约束没有超出限制；
- 工作量比较合适；
- 与其他代理的合作没有问题；
- 对将来其他工具购买的一些好的建议。

第七部分

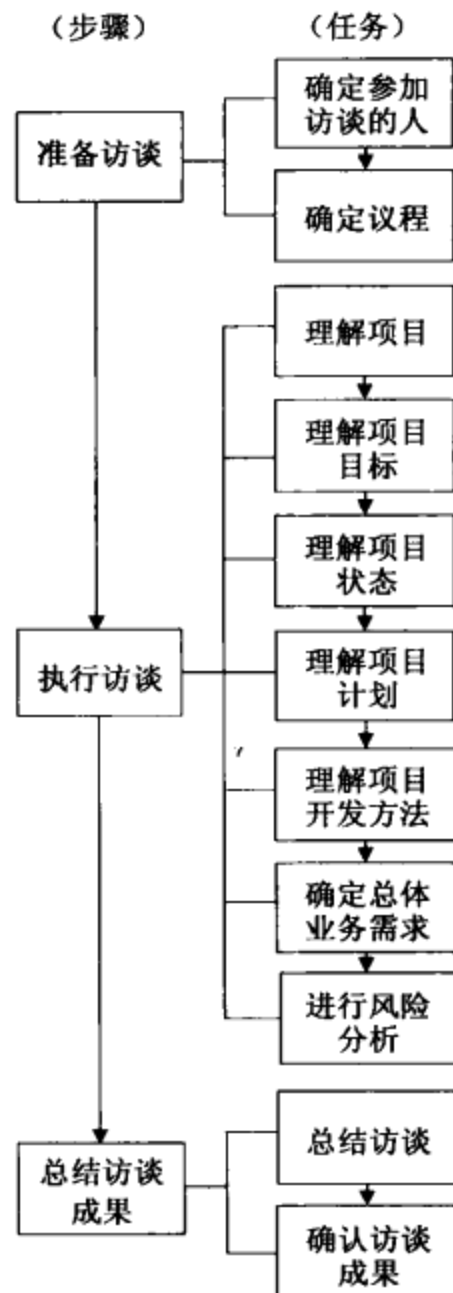
附 录

螺旋（敏捷）测试法

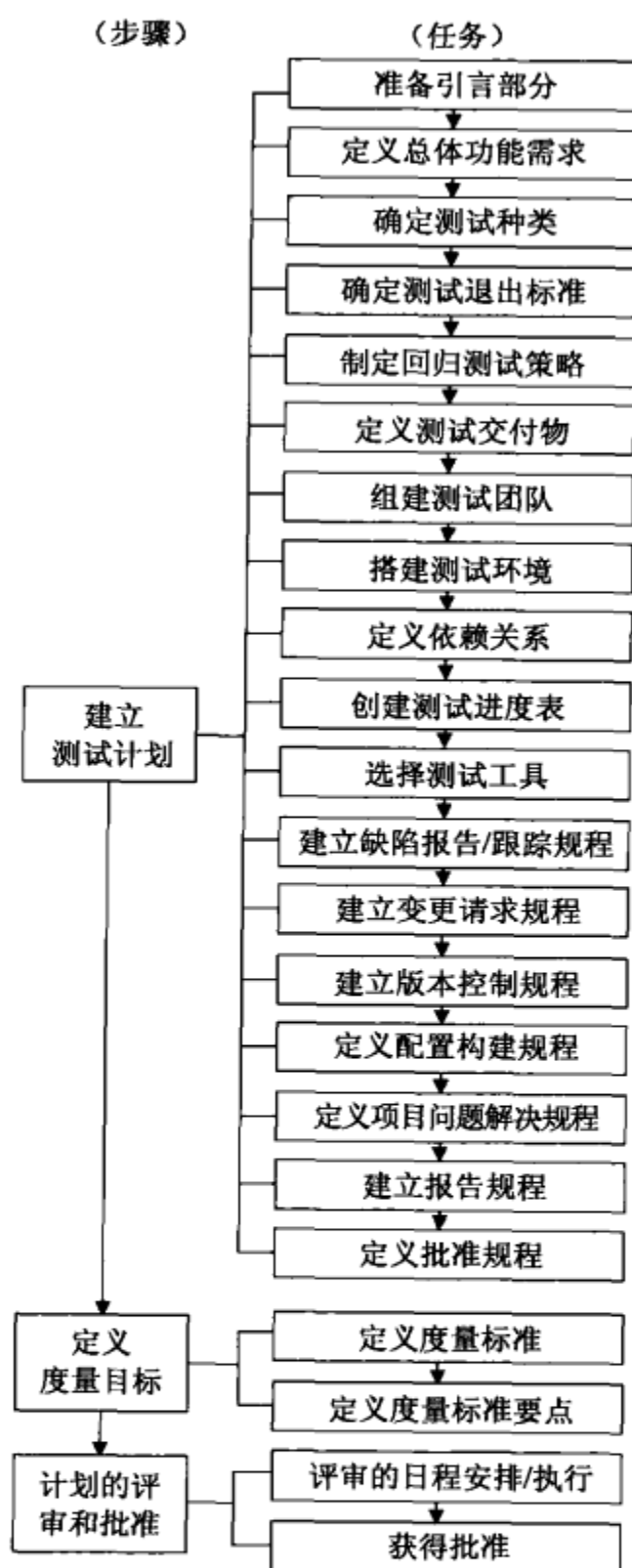
图A-1至图A-10是螺旋测试法的图形表示，由Deming的计划、执行、检查、改进（PDCA）质量轮、部分、步骤和任务组成。



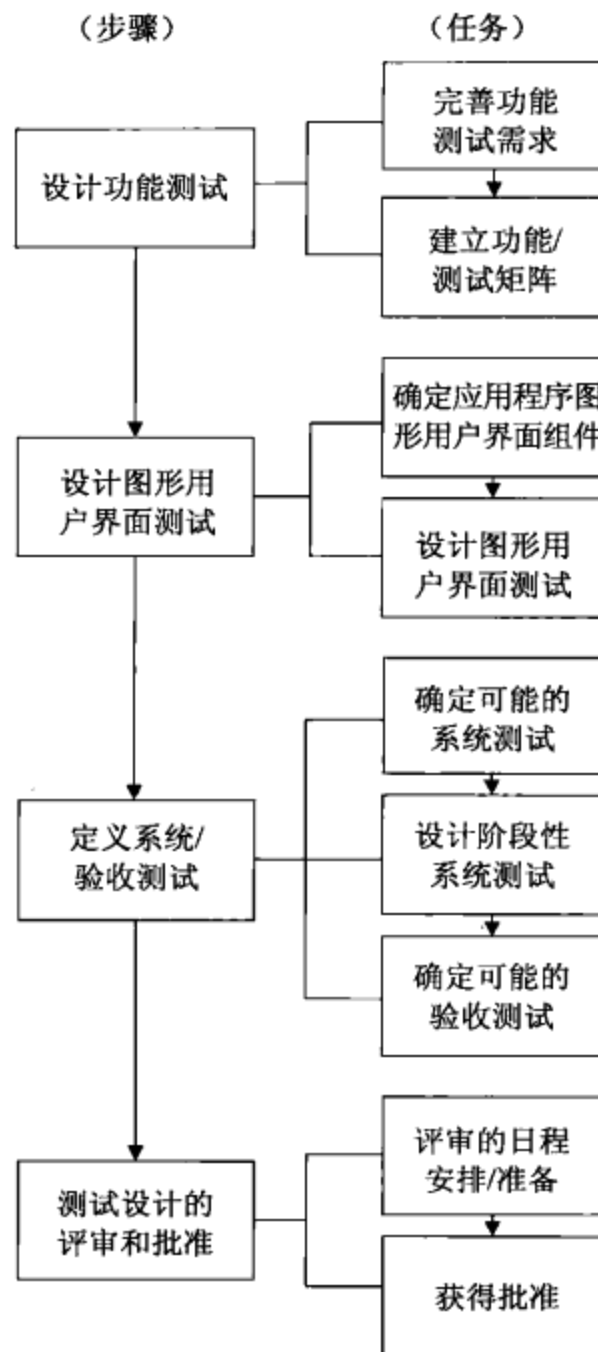
图A-1 持续改进



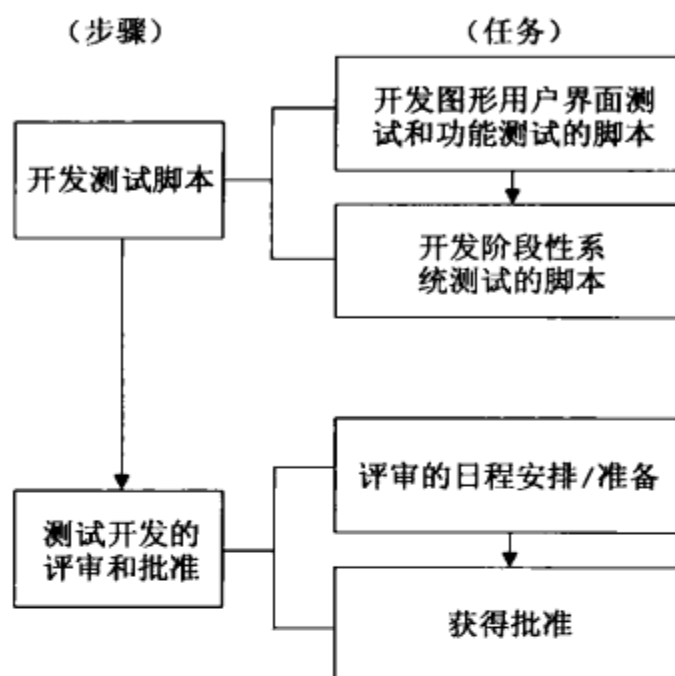
图A-2 信息收集



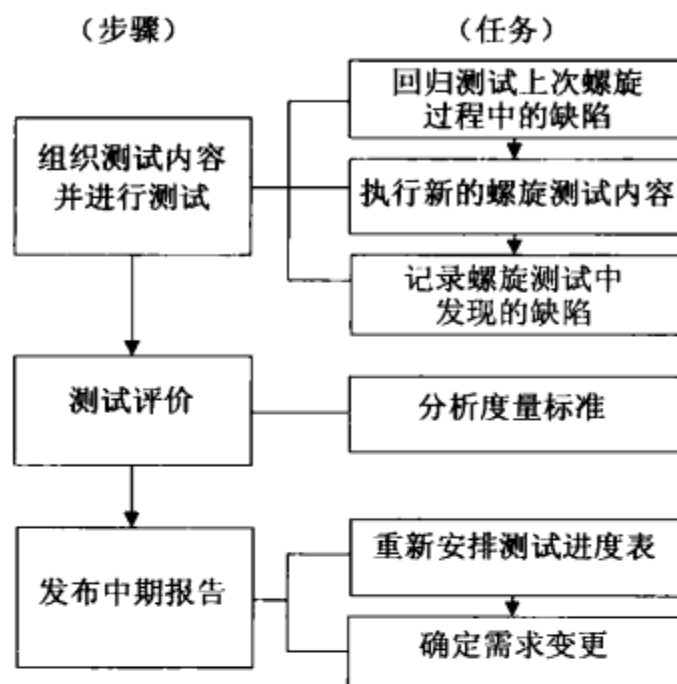
图A-3 测试计划



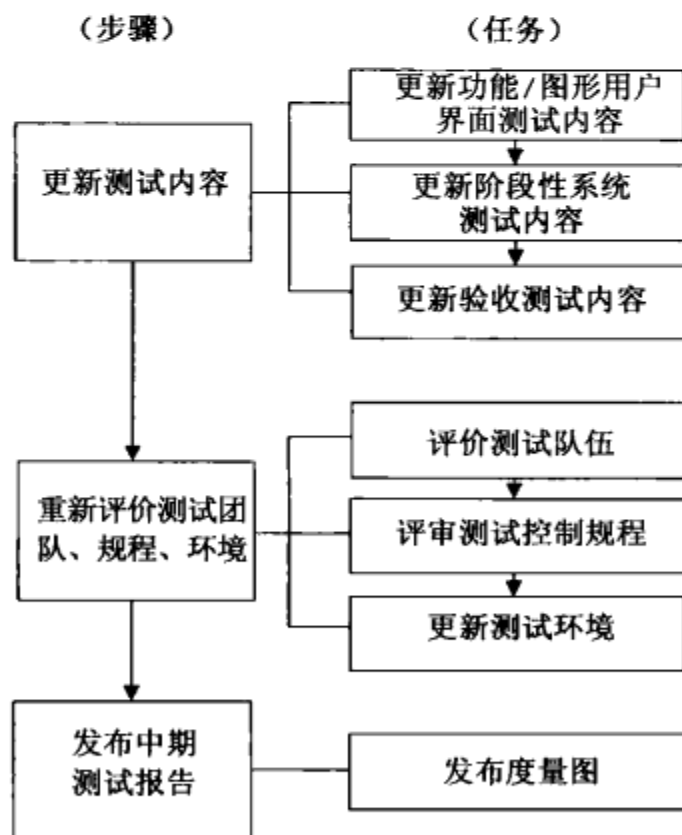
图A-4 测试用例设计



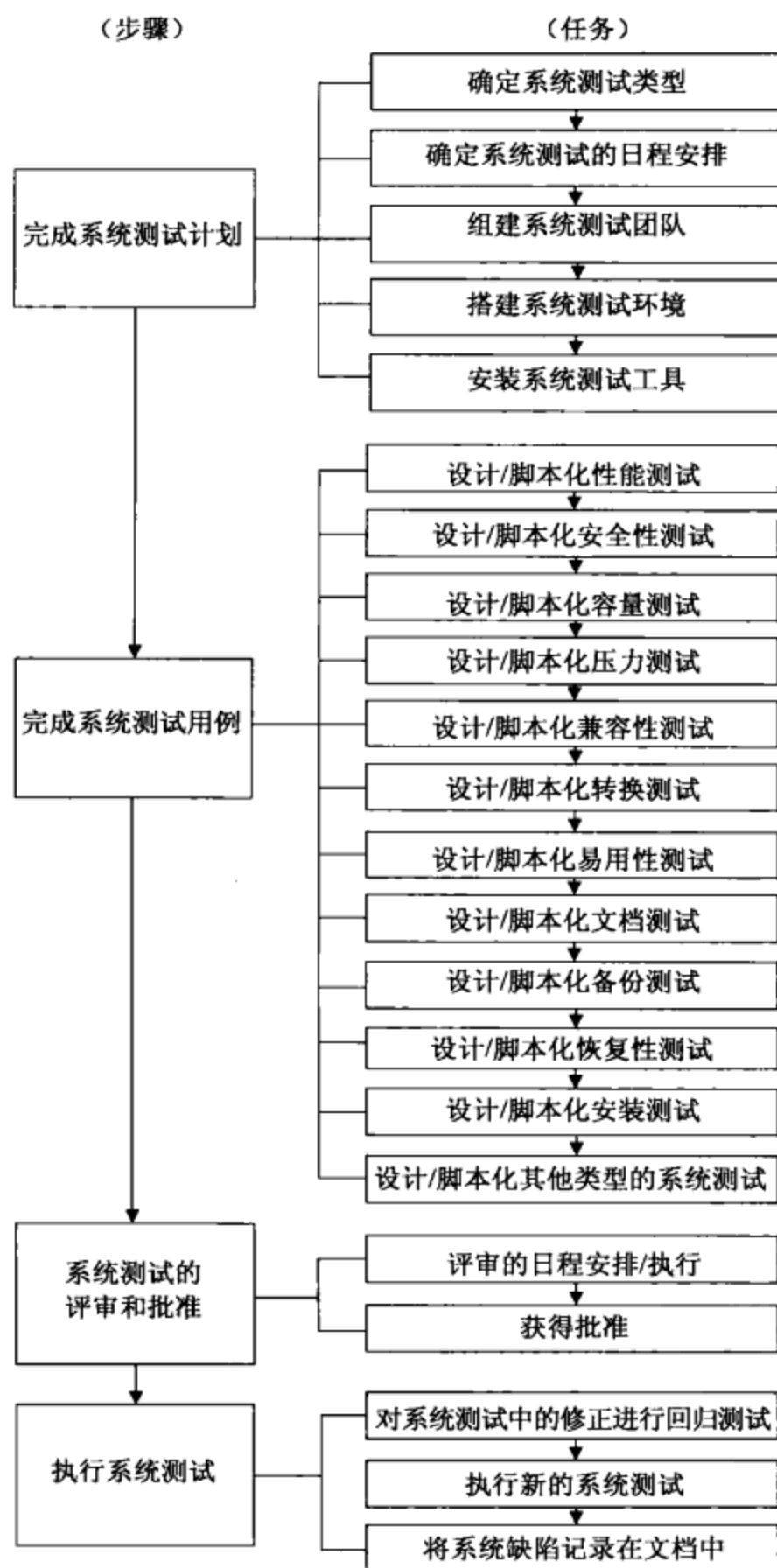
图A-5 测试开发



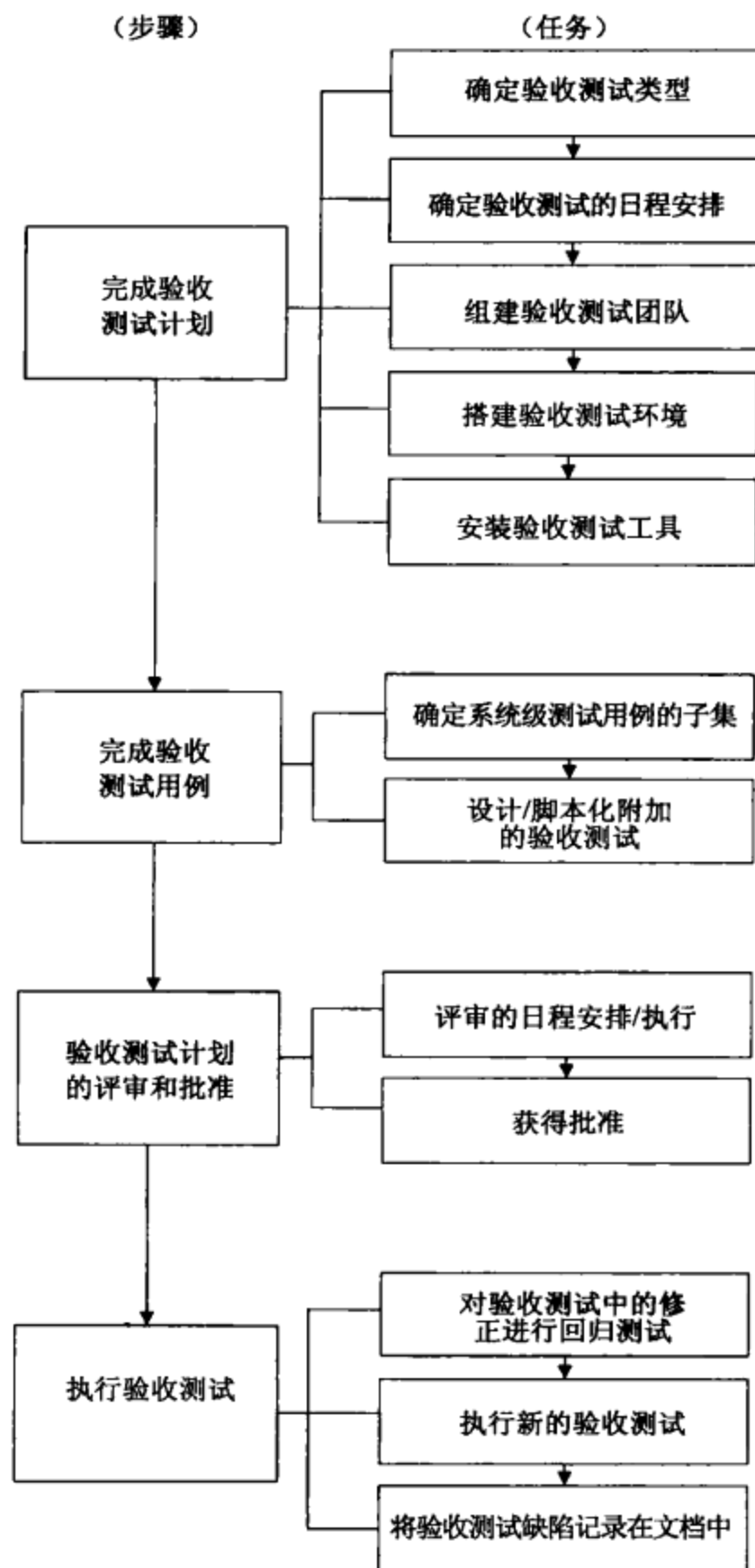
图A-6 测试执行/评价



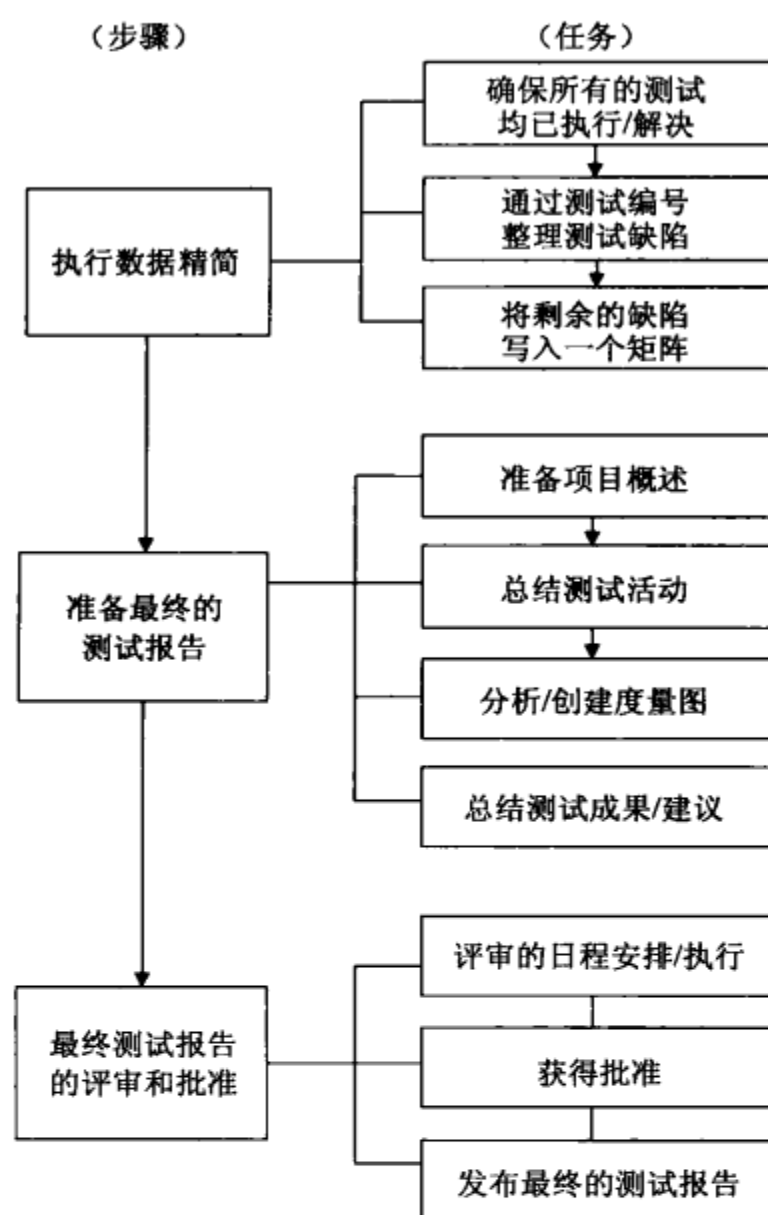
图A-7 准备下一次螺旋测试



图A-8 进行系统测试



图A-9 进行验收测试



图A-10 总结/报告螺旋测试结果

软件质量保证计划

本附录为应用项目提供了软件质量保证计划的样本。为了强调计划的一般理论和技术，此处忽略了项目细节。

1. 目标
2. 参考文档
 - 2.1 MIS标准
 - 2.2 MIS软件指导
 - 2.3 软件需求规约
 - 2.4 通用项目计划
 - 2.5 通用软件测试计划
 - 2.6 软件配置管理计划
3. 管理
 - 3.1 组织结构
 - 3.2 任务与责任
 - 3.2.1 项目主管
 - 3.2.2 软件开发团队
 - 3.2.3 测试小组
4. 文档
 - 4.1 软件需求规约
 - 4.2 系统用户手册
 - 4.3 安装指南
 - 4.4 测试结果总结
 - 4.5 软件单元文档
 - 4.5.1 概要设计文档
 - 4.5.2 详细设计文档
 - 4.5.3 其他文档
 - 4.6 翻译软件单元
5. 标准、实践及惯例
6. 评审和审查
7. 软件配置管理

- 8. 问题报告和修正
- 9. 工具、技术和方法
- 10. 代码控制
- 11. 介质控制
- 12. 供应商控制
- 13. 记录收集、维护和保存
- 14. 测试方法

需求规约

需求规约是针对特定环境下以特征来划分的实现某种功能的软件产品、程序或应用软件的说明书（来源：IEEE推荐使用的软件需求规约）。

1. 引言

- 1.1 目标
- 1.2 范围
- 1.3 定义、缩写、简写
- 1.4 参考资料
- 1.5 概述

2. 总体描述

2.1 产品概述

- 2.1.1 系统界面
- 2.1.2 用户界面
- 2.1.3 硬件及接口
- 2.1.4 软件接口
- 2.1.5 交互界面
- 2.1.6 内存限制
- 2.1.7 操作
- 2.1.8 网站需求

2.2 产品功能

2.3 用户特征

2.4 约束

2.5 假设和依赖

2.6 需求分配

3. 特殊需求

3.1 外部界面/接口需求

- 3.1.1 用户界面
- 3.1.2 硬件接口

- 3.1.3 软件接口
- 3.1.4 交互界面
- 3.2 系统特性
 - 3.2.1 系统特性1
 - 3.2.1.1 特性简介及目的
 - 3.2.1.2 请求/响应顺序
 - 3.2.1.3 相关的功能需求
 - 3.2.1.4 特征简介及目的
 - 3.2.1.5 请求/响应顺序
 - 3.2.1.6 相关的功能需求
 - 3.2.1.6.1 功能需求1
 - .
 - .
 - .
 - 3.2.1.6.*n* 功能需求*n*
 - 3.2.2 系统特性2
 - .
 - .
 - .
 - 3.2.*m* 系统特性*m*
 - .
 - .
 - .
- 3.3 性能需求
- 3.4 设计限制
- 3.5 软件系统属性
- 3.6 其他需求
- 4. 补充信息
 - 4.1 目录及索引
 - 4.2 附录

变更请求表

下面的变更请求表样例是记录和传播变更控制活动的文档。

变更请求表

报告编号：_____ 变更需求编号：_____

系统受到的影响：_____

子系统受到的影响：_____

文档受到的影响：_____

问题描述：

采取的操作：

E.1 单元测试计划

单元测试计划是基于程序或设计规约的，在正式的测试环境中使用。下面是单元测试计划内容表的样例。

1. 引言部分
 - a. 测试策略及方法
 - b. 测试范围
 - c. 测试假设
2. 走查（静态测试）
 - a. 发现及更正缺陷
 - b. 改进意见
 - c. 遵从结构化编程规范
 - d. 语言标准
 - e. 开发文档标准
3. 测试用例（动态测试）
 - a. 输入测试数据
 - b. 初始条件
 - c. 期望值
 - d. 测试日志状态
4. 环境需求
 - a. 测试策略及方法
 - b. 平台
 - c. 程序库
 - d. 工具
 - e. 测试规程
 - f. 状态报告

E.2 系统/验收测试计划

系统或验收测试计划是基于需求规约的，并且要求在正式的测试环境中使用。系统测试主要

对整个应用程序的功能及性能进行评价，是由多种测试组成的，包括性能测试、易用性测试、压力测试、文档测试、安全性测试、容量测试、可恢复性测试等。验收测试是由用户执行的测试，用以验证应用程序是否能够满足最初的业务目标和系统需求，通常是由系统测试的子集组成的。

1. 引言

- a. 系统描述（例如，对系统的简短描述）
- b. 目标（例如，测试计划的目标）
- c. 假设（例如，计算机可能工作的所有时间等）
- d. 风险（例如，单元测试没有完成的风险）
- e. 应急准备（例如，备份过程等）
- f. 约束条件（例如，资源短缺）
- g. 批准签字（例如，签署文件的权限）

2. 测试方法与策略

- a. 测试范围（例如，将要进行的测试）
- b. 测试方法（例如，测试工具、黑盒测试）
- c. 测试类型（例如，单元测试、系统测试、静态测试、动态测试、手工测试）
- d. 后勤（例如，定位、网站需求等）
- e. 回归测试的策略（例如，在每个build之间）
- f. 测试设备（例如，对需要进行测试的场所的一般描述）
- g. 测试规程（例如，缺陷修复验收、缺陷优先级等）
- h. 测试组织（例如，描述QA/测试小组）
- i. 测试库（例如，定位及描述）
- j. 测试工具（例如，捕获/回放回归测试工具）
- k. 版本控制（例如，控制不同版本的流程）
- l. 配置构建（例如，如何构建系统）
- m. 变更控制（例如，管理变更需求的流程）

3. 测试执行设置

- a. 系统测试规程（例如，开始标准、准备工作等）
- b. 设备（例如，测试环境、实验室的详细信息）
- c. 资源（例如，人员、培训、时间限制）
- d. 工具计划（例如，具体的工具、包、特殊的软件）
- e. 测试组织（例如，关于人员、角色、职责的详细信息）

4. 测试规约

- a. 功能分解（例如，要测试功能规约中的哪些功能）
- b. 不需要测试的功能（例如，超出范围的）
- c. 单元测试用例（例如，具体的单元测试用例）
- d. 集成测试用例（例如，具体的集成测试用例）

- e. 系统测试用例（例如，具体的系统测试用例）
- f. 验收测试用例（例如，具体的验收测试用例）
- 5. 测试规程
 - a. 测试用例、脚本、数据开发（例如，开发及维护流程）
 - b. 测试执行（例如，执行测试的流程）
 - c. 修正（例如，对已发现缺陷进行修正的流程）
 - d. 版本控制（例如，控制软件组件版本的流程）
 - e. 维护测试库
 - f. 自动化测试工具的使用（例如，工具标准）
 - g. 项目管理（例如，问题及缺陷的管理）
 - h. 监控及状态报告（例如，临时报告与总结报告）
- 6. 测试工具
 - a. 将要使用的工具（例如，具体的工具及性能）
 - b. 安装和设置（例如，使用说明书）
 - c. 支持和帮助（例如，供应商帮助）
- 7. 人力资源
 - a. 所需技能（例如，手动/自动化测试技能）
 - b. 角色和职责（例如，什么人什么时候做什么事情）
 - c. 数量和时间需求（例如，资源平衡）
 - d. 培训需求（例如，对人员进行工具培训）
- 8. 测试进度安排
 - a. 制定测试计划（例如，开始及结束时间）
 - b. 设计测试用例（例如，由测试类型决定的开始及结束时间）
 - c. 创建测试用例（例如，由测试类型决定的开始及结束时间）
 - d. 执行测试用例（例如，由测试类型决定的开始及结束时间）
 - e. 问题报告（例如，开始及结束时间）
 - f. 开发测试总结报告（例如，开始及结束时间）
 - g. 编写测试总结报告（例如，开始及结束时间）

E.3 需求可追溯性矩阵

需求可追溯性矩阵是一个从分析到实现阶段跟踪用户需求的文档。可以将其用于完整性检查，以验证所有的需求都存在，或者没有不必要的/多余的功能，并且可以将其作为新职员的维护指南。在开发周期的每一步骤中记录需求、代码及相应的测试用例，以保证最终系统能够满足用户需求。用户和开发人员都可以很容易地将需求与设计规约、程序及测试用例进行前后参照。

用户需求 参考号	系统需求 参考号	设计规约	编码组件 参考号	单元测试用例 参考号	集成测试用例 参考号	系统测试用例 参考号	验收测试用例 参考号
1.1客户 需有效	1.1.2在线 客户屏幕	客户界 面规约	CUS105, CUS217	CUS105.1.1, CUS2171.1	Int 1.25, Int 1.26	Sys4.75, Sys4.76	Acc2.25, Acc2.26
.
.
.

E.4 测试计划（客户/服务器架构和因特网螺旋测试）

客户/服务器架构的测试计划是以最初对开发人员的访谈中收集的信息以及项目进行中任何其他可用的信息为基础的。因为需求规约在螺旋式开发环境下可能不适用，所以这个测试计划就是“活文档（living document）”。经过每个螺旋测试，新的信息添加进来，旧的信息随着环境的变化而更新。主要的测试工作是进行功能测试、图形用户界面测试、系统测试、验收测试及回归测试。然而这些测试在某些特定的状态下不需要执行，或者可能同时进行。

测试计划的封面包括测试项目的标题、作者、当前版本号及最后变更日期。下一页包括项目发起人、开发经理、测试经理、质量保证经理及其他相关人员的联合签名。

下面是测试计划内容表的样例。

1. 引言

- 1.1 目标
- 1.2 管理摘要
- 1.3 项目文档
- 1.4 风险

2. 范围

- 2.1 在范围以内
- 2.2 测试需求
 - 2.2.1 总体功能需求
 - 2.2.2 用户业务/界面规则
- 2.3 图形用户界面测试
- 2.4 关键系统/验收测试
 - 2.4.1 性能测试
 - 2.4.2 安全性测试
 - 2.4.3 容量测试
 - 2.4.4 压力测试
 - 2.4.5 兼容性测试

- 2.4.6 转换测试
- 2.4.7 易用性测试
- 2.4.8 文档测试
- 2.4.9 备份测试
- 2.4.10 可恢复性测试
- 2.4.11 安装测试
- 2.5 回归测试
- 2.6 超出范围
- 3. 测试步骤
 - 3.1 总体测试结构
 - 3.2 数据
 - 3.3 界面
 - 3.4 环境/系统需求
 - 3.5 依赖条件
 - 3.6 回归测试策略
 - 3.7 缺陷跟踪及解决
 - 3.8 问题解决
 - 3.9 变更请求
 - 3.10 资源需求
 - 3.10.1 人员
 - 3.10.2 硬件
 - 3.10.3 测试环境
 - 3.11 里程碑/进度表
 - 3.12 软件配置管理
 - 3.13 测试提交
 - 3.14 测试工具
 - 3.15 度量标准
 - 3.16 测试进入/退出标准
 - 3.17 中期状态报告/总结状态报告
 - 3.18 批准

E.5 功能/测试矩阵

功能/测试矩阵将测试和功能关联起来。这个矩阵提供了证明测试策略完整性的方法，以图解的形式说明了哪些测试用例测试哪些功能。

业务功能	测试用例				
	1	2	3	4	5

E.6 图形用户界面组件测试矩阵（客户/服务器架构及因特网螺旋测试）

使用图形用户界面组件测试矩阵，每个图形用户界面组件可以由名称和图形用户界面类型定义及记录。进行图形用户界面测试时，依照预先定义的一组图形用户界面测试对每个组件进行测试。

名 称	图形用户界面类型					通过/失败	日 期	测 试 者
	窗 口	菜 单	表 单	图 标	控 件			
主窗口	√							
客户订单窗口	√							
编辑订单窗口	√							
菜单栏		√						
工具栏					√			
•								
•								
•								

464

E.7 基于图形用户界面的功能测试矩阵（客户/服务器架构及因特网螺旋测试）

下面是基于图形用户界面的功能测试矩阵模板，可以用来记录基于图形用户界面的测试用

例。包括功能和相关的图形用户界面对象或基本组件（窗口、菜单、格式、按钮及控件）。每个测试包括需求编号、测试目标、测试规程（步骤或脚本）、期望结果、是否通过测试、测试者和测试日期。同样可以用作测试用例日志。

功能（输入名称）

用例编号	需求编号	测试目标	用例步骤	期望结果	测试结果	测试者	日期
			1.				
			2.				
			3.				
图形用户界面对象（菜单、按钮、列表框等）							
			1.				
			2.				
			3.				
图形用户界面对象（菜单、按钮、列表框等）							
			1.				
			2.				
			3.				
图形用户界面对象（菜单、按钮、列表框等）							
			1.				
			2.				
			3.				
图形用户界面对象（菜单、按钮、列表框等）							
			1.				
			2.				
			3.				

E.8 测试用例

执行测试时可以按照下面测试用例所定义的循序渐进的流程进行，包括测试的目标和条件、建立测试的步骤、输入的数据、期望结果和实际结果。同时也包含软件、环境、版本、测试ID、

界面、测试类型等其他信息。

日期: _____	测试者: _____
系统: _____	环境: _____
目标: _____	测试ID: _____ 需求ID: _____
功能: _____	界面: _____
版本: _____	测试类型: _____

(单元测试、集成测试、系统测试、验收测试)

要测试的条件:

数据与要执行的步骤:

期望结果:

实际结果: 通过 ☐ 失败 ☐

E.9 测试用例日志

测试用例日志在测试期间为将要执行的测试类型记录测试用例。同时记录的测试结果为测试日志总结报告提供了详实的论据,并有助于在需要的时候重新进行测试。

测试名称：填写测试的名称

测试用例编写者：填写测试用例编写者名字

测试者：填写测试者名字

项目ID/名称：填写项目名称

测试周期ID：填写测试周期ID

测试完成日期：填写测试用例完成的日期

测试用例ID	测试目标ID	种 类	条 件	期望结果	实际结果	需求ID
输入ID	输入测试计划中的ID	输入测试种类 (编辑的、数字的、显示的等)	输入特殊的测试条件	记述执行此条件时的特殊期望值	记录“通过”或“失败”	输入特殊需求ID

E.10 测试日志总结报告

测试日志总结报告是为了报告状态及收集度量标准而记录测试者测试日志中的正在执行或已完成的测试用例。

468

完 成 者		填写做报告的测试者名字		报告日期		填写报告日期		
项目ID/名称		填写项目ID/名称		测试名称/事件		填写测试类型名称（单元测试、集成测试、系统测试、验收测试）		
测试用例总数		填写测试用例总数		测试子类型		填写测试子类型名称（界面测试、容量测试、压力测试、用户测试、平行测试）		
周/月	当前周期	迄今为止的项目情况	已开始测试的百分比	当前周期	已开放测试的百分比	当前周期	迄今为止的项目情况	已完成测试的百分比
填写测试周期	填写此周期内已开始的测试用例数	填写迄今为止开始的测试用例总数	开始的测试用例总数除以测试用例总数	填写此周期内已开始的测试用例数	开始的测试用例总数除以测试用例总数	填写此周期内已开始的测试用例数	填写迄今为止开始的测试用例总数	开始的测试用例总数除以测试用例总数
总数:								

E.11 系统总结报告

系统总结报告应该为每一个主要的测试事件做准备，有时还需要对所有的测试进行总结。下面是报告所要提供的信息要点。

1. 一般信息
 - 1.1 测试目标。要对包括已测试软件的一般功能和已执行的测试分析在内的测试目标进行总结。目标中包括功能、性能等。
 - 1.2 环境。要确定软件发起人、开发经理、用户组织以及软件安装的计算机中心等信息。要将测试环境与操作环境中的不同方式标注出来，对所有不同的结果进行评估。
 - 1.3 参考文献。要列出可用的参考资料，包括：
 - a. 项目授权；
 - b. 以前发布的项目文档；
 - c. 与项目相关的文档；
 - d. 标准及其他参考文档。
2. 测试结果及发现

每个测试的结果和发现要分别记录。

 - 2.1 测试（标出名称）。
 - 2.1.1 确认测试。本测试的数据输入、输出结果，包括内部生成数据的输出，要与数据输入、输出需求进行对比。发现的内容要一并记录。
 - 2.1.2 验证测试。要列出与期望结果的差异。
 - 2.2 测试（标出名称）。第2个以及后面的测试的结果和发现应按照前一段中格式表述。

3. 软件功能及发现

3.1 功能（标出名称）。

3.1.1 性能。对功能进行简要描述。对设计用来满足此功能的软件能力进行描述。要包括从验证能力的一个或多个测试中发现的内容。

3.1.2 限制。要确定已测试数据值的排列。对在软件中发现的在测试过程中与此功能相关的缺点、限制和约束进行标注。

3.2 功能（标出名称）。第2个及以后的功能的发现应按照与第3.1段相似的格式表述。

469

4. 分析总结

4.1 兼容性。描述测试对软件能力的验证。当测试对满足一个或多个特殊性能需求进行验证时，要准备关于这些需求的结果比较的发现要点。对测试环境与操作环境任何不同的结果都要进行评估。

4.2 缺陷。要列出测试中所发现的软件缺陷，并评估其对软件性能造成的影响。对所有发现的缺陷给性能造成的累积的或全部的影响进行总结。

4.3 图表分析。可以使用图片作为论证开发项目历史的材料，包括缺陷趋势分析、根源分析等（推荐使用项目简讯图片作为例证）。

4.4 风险。要列出当软件转化为产品时所面临的业务风险。

4.5 建议及评估。对于每个缺陷，要提供修正其所花费的时间和工作量以及以下建议：

- a. 每个修正的紧急程度；
- b. 负责修正的人员；
- c. 如何进行修正。

4.6 意见。要对产品的软件准备工作进行评估。

E.12 缺陷报告

缺陷报告用来记录测试过程中发现的异常情况，包括重现问题所需的所有信息、作者、发布/创建编号、打开/关闭日期、问题域、问题描述、测试环境、缺陷类型、如何发现的、发现者、优先级、严重性、状态等。

软件问题报告

缺陷ID：（必填）

由计算机生成

作者：（必填）

由计算机生成

发布/构建编号：（必填）

发现问题时的版本

打开日期：（必填）

由计算机生成

关闭日期：（必填）

470

QA关闭时由计算机生成

问题域：（必填）

如加法次序等

缺陷或增强：（必填）

缺陷（默认）

增强

问题名称：（必填）

简短的一行描述

问题描述：（必填）

对问题的详细描述，若可能附上屏幕截图

当前环境：（必填）

如Win95T/Oracle 4.0 NT

其他环境：

如Win95T/Oracle 4.0 NT

缺陷类型：（必填）

体系结构

连通性

一致性

数据库完整性

文件

功能性（默认）

图形用户界面

安装

内存

性能

安全及控制

标准及惯例

压力

易用性

发现者：（必填）

外部客户

内部客户

开发者

质量保证（默认）

如何发现：（必填）

评审

走查

JAD

测试（默认）

分配给：（必填）

优先级：（必填）

紧急

高（默认）

中

低

严重性：（必填）

严重

高（默认）

中

低

状态：（必填）

开放（默认）

正由开发部门评审

开发部门返回

准备在下一版本中测试

已关闭（QA）

（QA）返回

推迟至下一次发布

状态描述：

（当状态=“开发部门返回”，“准备在下一版本中测试”时使用）

修复者：

（当状态=“准备在下一版本中测试”时使用）

已计划的修复建立：

（当状态=“准备在下一版本中测试”时使用）

471

E.13 测试进度表

测试进度表包括测试步骤（或者和任务）、目标开始时间、目标结束时间和负责人。同时描述了怎样评审、跟踪及批准测试。

测试步骤	开始时间	结束时间	负责人
第一次螺旋			
信息收集			
准备访谈	xx/xx/xx	xx/xx/xx	
主持访谈	xx/xx/xx	xx/xx/xx	
总结访谈成果	xx/xx/xx	xx/xx/xx	

472

(续)

测试步骤	开始时间	结束时间	负责人
测试计划			
建立测试计划	xx/xx/xx	xx/xx/xx	
确定度量目标	xx/xx/xx	xx/xx/xx	
评审/批准计划	xx/xx/xx	xx/xx/xx	
设计测试用例			
设计功能测试	xx/xx/xx	xx/xx/xx	
设计图形用户界面测试	xx/xx/xx	xx/xx/xx	
确定系统/验收测试	xx/xx/xx	xx/xx/xx	
评审/批准设计	xx/xx/xx	xx/xx/xx	
测试开发			
开发测试脚本	xx/xx/xx	xx/xx/xx	
评审/批准测试开发	xx/xx/xx	xx/xx/xx	
测试执行/评估			
安装及测试	xx/xx/xx	xx/xx/xx	
评估	xx/xx/xx	xx/xx/xx	
为下一螺旋做准备			
改进测试	xx/xx/xx	xx/xx/xx	
重新评估测试团队、规程及测试环境	xx/xx/xx	xx/xx/xx	
发布中期报告	xx/xx/xx	xx/xx/xx	
•			
•			
•			
最后一次螺旋……			
测试执行/评估			
安装及测试	xx/xx/xx	xx/xx/xx	
评估	xx/xx/xx	xx/xx/xx	
•			
•			
•			
实施系统测试			
完成系统测试计划	xx/xx/xx	xx/xx/xx	
完成系统测试用例	xx/xx/xx	xx/xx/xx	
评审/批准系统测试	xx/xx/xx	xx/xx/xx	

(续)

测试步骤	开始时间	结束时间	负责人
执行系统测试	xx/xx/xx	xx/xx/xx	
实施验收测试			
完成验收测试计划	xx/xx/xx	xx/xx/xx	
完成验收测试用例	xx/xx/xx	xx/xx/xx	
评审/批准验收测试计划	xx/xx/xx	xx/xx/xx	
执行验收测试	xx/xx/xx	xx/xx/xx	
总结/报告螺旋测试结果			
进行数据归约处理	xx/xx/xx	xx/xx/xx	
准备最终测试报告	xx/xx/xx	xx/xx/xx	
评审/批准最终测试报告	xx/xx/xx	xx/xx/xx	

E.14 再测试矩阵

再测试矩阵是用来关联测试用例与功能（或程序单元）的工具。矩阵中的检查标记意味着当功能（或程序单元）因为改善或者修改而引起变更时测试用例需要重新执行。没有标记的则不需要重新测试。再测试矩阵要在第一个螺旋测试阶段建立，但需要后续阶段的维护。如果在开发螺旋阶段修改功能（或程序单元），那么需要创建现存的或新的测试用例并且在再测试矩阵中检查，为下一次螺旋做好准备。随着后续螺旋的进行，一些功能（或者程序单元）会是稳定的，近期没有修改。想要有选择地去掉检查标记，需要在测试螺旋之间进行。

474

业务功能	测试用例				
	1	2	3	4	5
订单处理					
创建新订单					
完成订单					
编辑订单					
删除订单					
客户处理					
创建新客户					
编辑客户					
删除客户					
财务处理					
接受客户付款					

(续)

业务功能	测试用例				
	1	2	3	4	5
存储付款					
支付供应商					
填写支票					
显示记录					
库存处理					
获得厂商产品					
维护库存					
处理退单					
审计库存					
调整产品价格					
报表					
创建订单报表					
创建应收账款报表					
创建应付账款报表					
创建库存报表					

475

E.15 螺旋测试总结报告（客户/服务器架构及因特网螺旋测试）

最终螺旋测试报告的目标是描述测试结果，不仅包括产品功能能否正常工作，还包括测试小组关于产品中应用程序性能的评估。

对于某些项目，有口头报告即可，但有些项目则需要正式报告。下面的样例是对上述两种情况的折中，提供了报告所需的基本信息，不需要反复的准备工作。

I. 项目概述

II. 测试工作

- A. 测试小组
- B. 测试环境
- C. 测试类型
- D. 测试进度表
- E. 测试工具

III. 度量分析图

IV. 发现/建议

E.16 会议纪要

下面的会议纪要表格是用来记录项目信息收集会话的结果及后续工作的内容。

会议目的		会议日期	
开始时间		结束时间	
与会人员			
分布列表			

476

重要讨论内容

讨论条目 #1	详细信息	备 注
讨论条目 #2	详细信息	备 注
讨论条目 #3	详细信息	备 注
讨论条目 #4	详细信息	备 注

477

动作条目

- a.
- b.

- c.
- d.
- e.
- f.
- g.
- h.
- i.
- j.
- k.

E.17 测试审批

测试审批矩阵是针对测试提交的正式文档管理批准。

提交批准

测试提交名称	批准状态	批准人	批准时间

478

E.18 测试执行计划

测试执行计划是用于制定执行阶段工作计划的。

项目名称: _____
项目代码: _____

任务编号	任务/子任务	计划日期		资源	测试用例/ 脚本总数	完成的测试 用例/脚本	备注
		开始时间	结束时间				

479

日期: _____

E.19 测试项目里程碑

测试项目里程碑矩阵用于确定及跟踪关键的测试事件。

里 程 碑	预期时间	实际时间

480

E.20 PDCA 测试进度表

PDCA测试进度表矩阵是用来计划及跟踪“计划、执行、检查、改进”阶段工作的。

测试步骤	开始时间	结束时间	负 责 人
收集信息			
测试计划			
测试用例设计			
测试开发			
测试执行/评估			
准备下一次测试迭代			
进行系统测试			

(续)

测试步骤	开始时间	结束时间	负责人
进行验收测试			
测试总结/项目结束			

E.21 测试策略

481 测试策略是用来记录项目中用到的所有测试方法的。

- 1. 引言
 - 1.1 项目概述
 - <项目简介包括对项目范围的简要描述。>
 - 1.2 关于客户
 - <客户业务与项目相关。>
 - 1.3 应用程序/系统概述
 - <对应用程序功能理解的简明而概括的说明，并对业务功能进行分解。>
- 2. 测试范围
 - <在本节中列出大致的应用范围。>
 - 2.1 测试目标
 - <与特殊需求相关的测试目标。>
 - 2.2 测试类型
 - <列出功能测试、非功能测试、验收测试、回归测试、性能测试等测试类型。>
 - 2.3 范围之内
 - <事务、报告、界面、业务功能等。>
 - 2.4 超出范围
 - <确定没有明确涵盖在测试中的内容。>
 - 2.5 假设
 - <测试假设与测试范围相结合。>
 - 2.6 基线文档
 - <基线文档列表，带有版本号的原型。>
- 3. 测试方案
 - 3.1 测试方法
 - 3.1.1 进入标准
 - <测试计划开始前需要制定的标准列表。>

3.1.2 测试计划方法

<在准备必要的测试件时所采用的方法，不包括手工测试或自动化测试脚本、创建测试数据的方法等。>

3.1.3 测试文档

<包括定义及目标的测试文档列表。>

3.1.4 测试执行方法

<对如何执行测试的描述。>

3.1.5 测试执行清单

<在开始执行测试之前，测试小组可用的条目列表。>

3.1.6 测试迭代

<计划执行的测试迭代次数、进入及退出标准、每次测试迭代的范围。>

3.1.7 缺陷管理

<整个缺陷管理流程，包括缺陷会议、缺陷解决方案等。>

3.1.8 缺陷跟踪及缺陷报告

<列出测试执行过程中所使用的缺陷跟踪记录说明及缺陷记录样例模板。>

3.1.9 缺陷的分类及缺陷生命周期

<对缺陷生命周期、不同的缺陷严重级别及缺陷种类的详细注释。>

3.1.10 缺陷会议

<详细的缺陷会议流程，包括与会人员、职责及会议举行的频率。>

3.1.11 退出标准

<退出测试执行的标准。>

4. 资源

4.1 项目所需技能

<执行项目所需的分析技能。>

4.2 培训进度

<已列好时间表的针对项目进行的培训。>

4.3 离岸

4.3.1 测试组成员

<列出测试小组成员及其在项目中的角色、参与项目的日期。>

4.3.2 角色及责任

4.4 现场

4.4.1 测试组成员

<列出测试小组成员及其在项目中的角色、参与项目的日期。>

4.4.2 角色及责任

4.5 客户

<客户或客户代表的角色及职责。>

4.6 测试基础结构

4.6.1 硬件

<列出执行测试所需的硬件设备。>

4.6.2 软件

<列出执行测试所需的软件。>

5. 项目组织及沟通

<项目组织图、评审周期及递交客户文件的签署。>

5.1 上升模型

<针对不同的议题和关注点，制定上升流程和上升的时间线。>

5.2 暂停及恢复标准

<列出使得测试工作暂停或恢复的情况。>

5.3 风险、应急准备及缓解计划

<项目存在的风险、针对已确定风险的应急准备和缓解计划。>

5.4 进度

5.4.1 里程碑

<针对项目不同阶段的总体进度表，带有明确的由一系列工作计划的里程碑。>

5.4.2 详细计划

<使用MS-Project做出的详细项目计划，包含确定的任务/子任务以及总体进度表中提到的各个里程碑阶段中要用到的资源。>

5.4.3 交付物

<列出与测试文档中的项目相关的交付物，使交付物获得客户验收的机制。>

6. 附录

<在此记录前面所有章节中提到的附录。>

483

E.22 澄清请求

澄清请求矩阵用于记录测试人员在分析需求时可能出现的问题。

项目名称: _____

项目代码: _____

问题编号	参考文档	应用程序/ 功能	发现日期	澄清请求	发现者	状 态	回 应

484

(续)

问题编号	参考文档	应用程序/ 功能	发现日期	澄清请求	发现者	状态	回 应

E.23 屏幕数据映射

屏幕数据映射矩阵用来记录屏幕数据属性。

条目编号	测试用例ID	界面参考 (可选)	域 名	需要的数据	数据类型	数据形式	备 注
1	TS-01		账号	aabbcc	字母		
2	TS-01		账号	10099	数字	#####	
3	TS-05		同日期	31101999	日期	dd-mm-yyyy	

E.24 测试条件与测试用例

测试条件与测试用例矩阵用于将需求与映射到一个或多个测试用例的每个条件相关联。

条目编号	需求详细信息/来源 (功能规约/业务需求/其他)	条件编号	测试条件	测试用例编号

(续)

条目编号	需求详细信息/来源 (功能规约/业务需求/其他)	条件编号	测试条件	测试用例编号

E.25 项目状态报告

项目状态报告是为所有关键流程区域报告正在进行测试项目的状态。

目的：此模板将与项目相关的QA工作合并到所有关键流程区域中。每周发布给所有项目利益相关人。

486

项目名称 _____ 项目代码 _____

项目开始时间 _____ 项目经理 _____

项目阶段 _____ 周数及日期 _____

分布 _____

关键工作

	详细信息	注 解
提交内容		
决议		

本周进展

条 目	关键进展	工作/里程碑, 交付物	计划时间		实际时间		状态/注解	所 有 人
			开始时间	结束时间	开始时间	结束时间		

未计划工作

条 目	工 作	开始时间	结束时间	工作量(人小时)	备 注

下周计划工作

条 目	工 作	开始时间	结束时间	工作量(人小时)	备 注
计划但未完成的					
变更请求(新增)					
变更请求(未完成)					
问题(新增)					
问题(未完成)					

487

488

E.26 测试缺陷详细报告

测试缺陷详细报告是为所有关键过程区域报告正在测试的项目的缺陷状态的详细信息。

489

条目 编号	缺陷 ID	脚本 ID	测试用例描述	期望结果	实际结果	发现者	缺陷 状态	严重性	优先级	报告 日期	收尾 日期
1			不显示登 录页面	应显示登 录页面	不显示登 录页面	Bill	打开	3	1		
2			不能使用 文字框	应能使用文 字框		Joe	已关闭	2	2		
3			用户不能输入 数值	应允许用户 输入数值		Sam	已修改	3	3		
4			区域3没有显示 在列表中	模块3应显示 在列表中		Sally	已关闭	2	4		
5			显示错误信息	不应显示错 误信息		June	打开	2	5		

E.27 缺陷报告

缺陷报告用于报告具体缺陷的详细信息。

缺陷ID	日期	测试 脚本	测试 用例	期望 结果	实际 结果	状态	严重性	缺陷 类型	测试人 员备注	开发人 员备注	客户 备注

E.28 测试执行追踪管理表

测试执行追踪管理表是一个Excel表，提供了对通过或失败的测试用例数的全面的测试周期

视图、程序中发现的缺陷数目、缺陷状态、完成率以及由缺陷类型决定的缺陷严重性。

490

E.29 最终测试总结报告

最终测试总结报告是含有关键发现问题的测试项目的最终报告。

1. 引言

1.1 管理摘要

<突出项目的进度表、规模和缺陷数目,以及发生在项目生命周期中对管理有益的重要事件。>

1.2 项目概述

<本节涵盖客户业务和项目概述。>

1.3 测试范围

<关于测试范围及其详细信息。>

2. 测试方法

2.1 测试文档

<简要说明测试文档。>

2.2 测试迭代

<已实施的测试迭代的详细信息。>

2.3 缺陷管理

<简要说明执行过程中的缺陷管理流程。>

3. 测量

3.1 可追踪矩阵

<从需求到脚本的详细追踪信息。>

3.2 计划与实际

<计划与实际进度的详细信息,带有变化的原因。>

3.3 测试脚本总结

<测试执行后期的最终测试脚本总结。>

3.4 未测试/无效的功能

<关于未测试、无效或未提交的脚本及其原因的详细信息。>

4. 缺陷发现情况

4.1 最终缺陷总结

<在测试执行的后期对缺陷进行总结。>

4.2 延期的缺陷

<关于失败的测试用例以及说明了缺陷延期原因的处于延期状态的测试用例的详细信息。>

491

5. 分析

5.1 按种类划分的缺陷

<应生成图表以显示按种类划分的缺陷数目。>

5.2 按状态划分的缺陷

<应生成图表以显示按状态划分的缺陷数目。>

5.3 按严重级别划分的缺陷

<应生成图表以显示按严重级别划分的缺陷数目。>

5.4 问题

<项目进行过程中出现的问题的详细信息，需要记录并在管理过程中强调。>

5.5 风险

<对报告的缺陷要进行分析，并对任何可能会影响到业务的风险进行预测。>

5.6 观查项

<不能归类为问题和风险的其他重大事件。>

6. 测试小组

<参与项目的各方所有人员的姓名及角色。>

7. 附录

<在此记录前面所有章节中提到的附录。>

E.30 测试自动化策略

下面是测试自动化策略的标准格式，它将根据测试需求定制。

- 项目概述。
- 自动化的目的和目标。
- 自动化的范围——包含什么和排除什么。
- 自动化方法。
- 测试环境。
- 使用的工具——脚本和测试管理。
- 脚本命名约定。
- 资源与时间表。
- 培训需求。
- 风险和化解。
- 假设与约束。
- 进入和退出标准。
- 验收标准。
- 交付物。

检查表是一个强大的质量控制测试工具，之所以强大是因为其使用统计的方法区分了两种极端情况。检查表可用于在问题识别、原因分析时进行资料收集，或者在实现解决方案的过程中检查流程。

记录观察结果或条件时，在列表中相对应的条目处键入/不键入检查标记。用这种方法采集的信息仅限于简单地回答是或不是。检查表也可以量化或计算为后续计算及分析输入的数据。

F.1 需求阶段缺陷检查表

下面的需求阶段缺陷检查表用于为系统确定功能需求及规范。“遗漏”栏记录的是未包含的项，“错误”栏记录的是使用不当的项，“额外”栏记录的是最初没有定义但已被发现的项，“总数”栏填写的是遗漏及额外项的总和。

缺陷种类	遗 漏	错 误	额 外	总 数
1. 业务规则（或信息）不充分或部分遗漏。				
2. 性能标准（或信息）不充分或部分遗漏。				
3. 环境信息不充分或部分遗漏。				
4. 系统任务信息不充分或部分遗漏。				
5. 需求不兼容。				
6. 需求不完整。				
7. 需求有遗漏。				
8. 需求不正确。				
9. 规定的精确度不符合实际需要。				
10. 数据环境描述不充分。				
11. 外部界面定义错误。				
12. 没有充分考虑用户培训。				
13. 没有考虑到系统状态初始化。				
14. 没有充分定义功能。				
15. 用户需求的陈述不充分。				
16. 没有充分规定质量度量，如可维护性、可传输性等。				

F.2 逻辑设计阶段缺陷检查表

下面的逻辑设计阶段检查表用于验证系统的逻辑设计。“遗漏”栏记录的是未包含的项，“错误”栏记录的是使用不当的项，“额外”栏记录的是最初没有定义但已被发现的项，“总数”栏填写的是遗漏及额外项的总和。

494

缺陷种类	遗 漏	错 误	额 外	总 数
1. 数据定义不充分。				
2. 实体定义不完整。				
3. 实体基数不正确。				
4. 实体属性不完整。				
5. 违反规范化要求。				
6. 不正确的主键。				
7. 不正确的外键。				
8. 不正确的组合键。				
9. 不正确的实体子类型。				
10. 进程定义不充分。				
11. 父进程未完成。				
12. 子进程未完成。				
13. 进程输入/输出不正确。				
14. 基本进程定义不正确。				
15. 手动排除进程问题。				
16. 并发连接问题。				
17. 事件触发的进程定义错误。				
18. 错误的实体/进程创建关联。				
19. 错误的实体/进程读取关联。				
20. 错误的实体/进程更新关联。				
21. 错误的实体/进程删除关联。				

495

F.3 物理设计阶段缺陷检查表

下面的物理设计阶段检查表用于验证系统的物理设计。“遗漏”栏记录的是未包含的项，“错误”栏记录的是使用不当的项，“额外”栏记录的是最初没有定义，但已被发现的项，“总数”栏填写的是遗漏及额外项的总和。

缺陷种类	遗 漏	错 误	额 外	总 数
1. 逻辑或次序有错误。				
2. 处理不准确。				
3. 例程没有输入或输出所需的参数。				

(续)

缺陷种类	遗 漏	错 误	额 外	总 数
4. 例程没有接受在容许范围内的所有数据。				
5. 对输入数据进行限制及有效性检查。				
6. 恢复流程没有实现或不充分。				
7. 所需进程缺失或不充分。				
8. 值有错误的或不明确。				
9. 数据存储有错误或不充分。				
10. 变量缺失。				
11. 设计需求被不正确或错误解读。				
12. 数据库与数据环境不兼容。				
13. 分解的模块表现出模块间高依赖性。				
14. 没有对主要算法的精确度和速度进行评估。				
15. 控制结构不可扩展。				
16. 控制结构忽略了处理的优先级。				
17. 接口协议不正确。				
18. 实现算法的逻辑不正确。				
19. 没有将数据转换为正确格式。				
20. 没有考虑舍入和截断带来的影响。				
21. 没有对索引进行检查。				
22. 允许无限循环。				
23. 模块规约被错误解读。				
24. 违反数据库规则。				
25. 对于所有情况, 逻辑不完整。				
26. 忽略了特殊情况。				
27. 缺乏错误处理。				
28. 忽略了对时序的考虑。				
29. 需求规约在各个软件模块中分配不当。				
30. 接口规约被错误解读或错误实现。				
31. 系统在功能方面达标, 但性能没有达到要求。				
32. 软件的复杂程度不够, 无法解决存在的问题。				
33. 没有对算术上溢及下溢进行正确处理。				
34. 给定输入值的响应活动不适当或丢失。				
35. 输入某些特定值时, 算法近似值提供的精确度不够或者导致错误结果。				
36. 解决特殊问题时, 详细的逻辑规划出现错误。				

496

497

(续)

缺陷种类	遗 漏	错 误	额 外	总 数
37. 异常或临界值的输入可能会导致意想不到的结果，在代码内部无法正确解决。				
38. 算法的效率低下或者不能像其他更有效的算法那样快速地计算出结果。				
39. 算法没有覆盖所有可能出现的情况。				
40. 算法错误或导致错误的解决方案。				
41. 存在逻辑错误。				
42. 存在设计疏忽。				

F.4 程序单元设计阶段缺陷检查表

下面的程序单元设计阶段检查表用于验证系统的单元设计。“遗漏”栏记录的是未包含的项，“错误”栏记录的是使用不当的项，“额外”栏记录的是最初没有定义但已被发现的项，“总数”栏填写的是遗漏及多余项的总和。

缺陷种类	遗 漏	错 误	额 外	总 数
1. “if-then-else”结构的使用不正确？				
2. “do-while”结构的使用不正确？				
3. “do-until”结构的使用不正确？				
4. “case”结构的使用不正确？				
5. 存在死循环吗？				
6. 程序正常吗？				
7. 有“goto”语句吗？				
8. 程序可读吗？				
9. 程序高效吗？				
10. “case”结构能覆盖所有条件吗？				
11. 有无用代码吗？				
12. 程序含有可自我修复的代码吗？				
13. 算法表达式过于简单吗？				
14. 算法表达式过于复杂吗？				
15. 嵌套过深吗？				
16. 存在负的布尔逻辑吗？				
17. 存在混合的布尔逻辑吗？				
18. 可以跳入/跳出循环吗？				

F.5 代码编写阶段缺陷检查表

下面的代码阶段缺陷检查表用于验证从设计规约到可执行代码的转换。“遗漏”栏记录的是未包含的项，“错误”栏记录的是使用不当的项，“额外”栏记录的是最初没有定义但已被发现的项，“总数”栏所填写的是遗漏及额外项的总和。

499

缺陷种类	遗 漏	错 误	额 外	总 数
1. 判断逻辑/顺序错误或不充分。				
2. 数学计算错误或不充分。				
3. 分支错误。				
4. 分支或其他测试执行错误。				
5. 存在未定义的循环终止条件。				
6. 违反编程语言规则。				
7. 违反编程标准。				
8. 程序员错误解读语言结构。				
9. 存在印刷错误。				
10. 存在内存分配错误。				
11. 迭代方案没有成功。				
12. 存在I/O格式错误。				
13. 参数或脚本违反规则。				
14. 子程序的调用违反规则。				
15. 存在数据错误。				
16. 子程序没有结束语句。				
17. 准备或处理输入数据时出现错误。				
18. 磁带处理错误。				
19. 磁盘处理错误。				
20. 输出处理错误。				
21. 错误信息处理错误。				
22. 软件接口错误。				
23. 数据库接口错误。				
24. 用户界面错误。				
25. 索引和脚本错误。				
26. 迭代出现错误。				
27. 二进制处理出现错误。				
28. 语法错误。				
29. 初始化错误。				

500

(续)

缺陷种类	遗 漏	错 误	额 外	总 数
30. 参数使用混乱。				
31. 循环计数出现错误。				
32. 错误处理决策结果。				
33. 变量拥有多个名字或没有定义。				
34. 变量名拼写出现错误。				
35. 错误声明变量类型或维度。				
36. 库程序名混乱。				
37. 错误解析外部符号。				
38. 编译器出错。				
39. 存在异常或外部点。				
40. 存在浮点数下溢错误。				
41. 存在浮点数溢出错误。				
42. 允许浮点数或整数被零除。				
43. 排序错误。				
44. 在实时系统中保存及恢复正确的注册信息时失败。				
45. 连接硬件系统的软件接口错误。				

F.6 字段测试检查表

下面的字段测试仅限于特殊的字段或数据元素,并且可以用于验证是否正确地执行了与这个特殊字段相关的所有处理过程。

条 目	是	否	N/A	备 注
1. 是否验证了所有代码?				
2. 字段是否能完全更新?				
3. 字段是否足够大到可以采集总数?				
4. 是否在程序中正确地描述了字段?				
5. 字段是否能正确地初始化?				
6. 所有与字段相关的内容都使用了正确的字段名?				
7. 如果字段的内容受限,限制条件是否经过验证?				
8. 是否为鉴定及处理非法字段数据建立了规则?(如果没有,必须为错误处理事务类型开发数据。如果有,必须为非法字段数据验证规约处理准备测试条件。)				
9. 测试条件是否包含了大范围的典型有效处理值?				
10. 对于数值字段,是否测试了上限值和下限值?				

(续)

条 目	是	否	N/A	备 注
11. 对于数值字段, 是否测试了“零”值?				
12. 是否为数值字段准备了负数测试条件?				
13. 是否为字母字段准备了空条件?				
14. 对于字母或字母数值字段, 是否准备了比字段长度长的测试条件以检查截断处理?				
15. 是否输出了基于数据字典进行测试的所有有效条件?				
16. 是否检查了系统规约以确定所有的有效条件是否都被测试?				
17. 数据元素的所有者是否知道所有的有效条件是否经过测试?				
18. 数据元素的所有者是否报告了结果?				

F.7 记录测试检查表

下面的记录测试验证了可以正确地创建、输入、处理、存储和输出记录。

条 目	是	否	N/A	备 注
1. 是否为测试对第一条记录的处理准备好了条件?				
2. 是否为验证对最后一条记录的处理确定了条件?				
3. 是否正确处理了每个事务的所有多重记录?				
4. 是否正确处理了存储介质上的所有多重记录(例如, 永久文件或临时文件)?				
5. 是否测试了记录规模中的所有变量?				
6. 是否可以为每个记录类型检查作业控制语言?				
7. 是否可以处理拥有同一标识符的两个记录(例如, 同一应收账款文件的两笔付款)?				
8. 是否可以检索存储文件上的第一条记录?				
9. 是否可以检索存储文件上的最后一条记录?				
10. 所有输入的记录是否能够正确存储?				
11. 是否能够检索所有存储的记录?				
12. 相互连接的模块是否对于每个记录类型具有相同标识符?				
13. 数据输入功能是否准备了来自数据输入文件的正确记录?				
14. 用户文档对用户是否有帮助?				
15. 单一模块记录描述是否符合系统记录描述?				
16. 记录的存储定义是否符合记录的系统定义?				
17. 记录描述是否通常贯穿整个软件系统?				
18. 现有的记录格式是否与其他系统创建的文件所使用的格式相符?				

503

504

F.8 文件测试检查表

下面的文件测试验证了所需的所有文件都包含在了正在测试的系统中，并在操作基础设施中正确地记录，而且这些文件与需要文件数据的软件组件正确地连接。

条 目	是	否	N/A	备 注
1. 是否为每个文件都准备好了测试环境？				
2. 是否为带有模块的每个文件接口都准备好了测试环境？				
3. 是否为验证每个作业控制条件准备好了测试环境（或者没有JCL环境中的设备）？				
4. 是否为验证即将使用的每个文件的正确版本准备好了测试环境？				
5. 是否为验证文件中的记录完整返回准备好了测试环境？				
6. 是否为在处理了文件中的最后一条记录之后正确地关闭每个文件做好了测试准备？				
7. 是否为完整地处理从系统开始到结束的每个记录类型做好了测试准备？				
8. 是否为通过系统处理的所有已输入的记录准备好了测试环境？				
9. 是否为在流程结束时正确地关闭已配置好但还没有使用过的文件准备好了测试环境？				
10. 是否为没有优先记录时创建文件准备好了测试环境？				
11. 是否为验证当所有文件中的记录都已删除时正确地关闭文件准备好了测试环境？				
12. 是否为验证所有作业控制语句的正确性准备好了测试环境？				

505

F.9 错误测试检查表

下面的错误测试确定了数据元素、数据元素之间的关系、记录和文件的关系以及逻辑处理条件中的错误。

条 目	是	否	N/A	备 注
1. 是由头脑风暴会议（包括最终用户/客户）确定的功能错误吗？				
2. 是由头脑风暴会议（包括项目人员）确定的结构错误状况吗？				
3. 对于下列状况是否确定了功能错误状况？				
拒绝无效代码				
拒绝超出范围的值				
拒绝错误数据关系				
拒绝无效数据				
拒绝下列未认证的事务：				

506

(续)

条 目	是	否	N/A	备 注
● 无效值				
● 无效客户				
● 无效产品				
● 无效事务类型				
● 无效价格				
数值字段中的字母数据				
数值字段中的空格				
数值字段中的全空白的情况				
正值字段中的负值				
负值字段中的正值				
字母字段中的数字				
字母字段中的空格				
值的长度超过字段的限制				
总数超过总数字段中的最大值				
总数的正确累积（多级总数的所有级别）				
不完整的事务（例如，丢失一个或多个字段）				
字段中的过期数据（例如，之前有效的代码失效）				
可接受的新值现在不可接受（例如，为还没有建立的行政区域建立新的行政代码）				
延迟事务				
值的变化将会影响某种关系（例如，如果使用单元数字去控制年份，那么89中的9转换成90中的0时，是否还能进行处理？）				
4. 字段规约的数据字典列表能否生成无效说明？				
5. 是否测试了下列架构性错误？				
页溢出				
报告格式与设计规划一致				
记录数据以修正报告部分				
印刷错误信息代表实际错误状况				
执行所有指令				
执行所有路径				
所有内部表格				
所有循环				
所有PERFORM类型例程				
所有编译器警告消息				
程序的正确版本				
在系统的任意部分变更之后其他未变更的部分重新生效				

F.10 使用测试检查表

508

下面的使用测试对最终用户使用系统的能力进行测试, 包括对系统输出及输出能力引导正确操作的理解能力。

条 目	是	否	N/A	备 注
1. 是否确定了所有最终用户的操作?				
2. 是否确定了足够的细节以确定这些操作涉及的信息系统输出条目的贡献?				
3. 是否确定了采取操作所使用的所有信息及涉及的操作?				
4. 受制于与特殊操作相关的测试的系统输出?				
5. 最终用户是否正确理解了输出报告和屏幕信息?				
6. 最终用户是否理解了逻辑类型及产生输出的运算?				
7. 最终用户是否确定了输出对于所采取操作的贡献?				
8. 最终用户是否能确定所采取操作的正确性?				
9. 如果不能, 那么其他人员能否确定所采取操作的正确性?				
10. 系统输出与定义的业务操作之间的关系?				
11. 矩阵的说明是否暗示了最终用户没有足够的信息以采取操作?				
12. 矩阵的分析是否暗示了最终用户进行了异常数量的不正确操作?				
13. 如果是, 最终用户是否愿意为了消除错误而修改系统以提供更好的信息?				

509

F.11 检索测试检查表

下面的检索测试验证了记录、字段及其他变量的位置, 并有助于确认搜索逻辑的正确性。

条 目	是	否	N/A	备 注
1. 是否确认了所有内部表格?				
2. 是否确认了所有出错消息的内部列表?				
3. 是否确认了所有内部逻辑路径(有多重选择时)?				
4. 是否确认了搜索逻辑?(有些时候, 算法用于确认所需实体。)				
5. 是否确认了所有认证例程?				
6. 是否确认了所有密码例程?				
7. 是否确认了所有业务处理逻辑所需的搜索(例如, 需要查找客户记录的逻辑)?				
8. 是否确认了数据库搜索例程?				
9. 是否确认了子系统搜索(例如, 在销售税子系统中查找税率)?				

(续)

条 目	是	否	N/A	备 注
10. 是否确认了复杂搜索逻辑（例如，搜索存在了90天以上并有100美元存款的账户，需要两个以上条件或者两条以上记录的逻辑）？				
11. 是否确认了处理模块的搜索程序？				
12. 是否为所有之前的搜索条件确定测试等级？				
13. 是否与最终用户进行了沟通以决定将来可能遇到的一次搜索类型？				
14. 如果是，能否在合理的投入内进行这些搜索（由项目组确定）？				
15. 如果不是，是否告知了最终用户，执行这些搜索或者重建系统以满足这些需求的成本有多少？				

510

F.12 匹配/合并检查表

下面的匹配/合并测试确保正确处理了所有合并及匹配的结合。这个测试通常包括两个或多个文件：输入事务及一个或多个文件，或者输入事务及内部表格。

条 目	是	否	N/A	备 注
1. 是否确定了所有与应用程序相关的文件？（在这个事务中，文件包括专用文件、数据库以及用于匹配和合并记录的内部分组。）				
2. 是否处理了下列匹配/合并条件？				
两个不同标识符记录的匹配/合并（例如，在工资单中插入新员工）				
没有记录的匹配/合并文件的匹配/合并				
匹配/合并记录是文件中的最低值的匹配/合并				
匹配/合并记录是文件中的最高值的匹配/合并				
匹配/合并记录与文件中的条目具有相同值的匹配/合并（例如，当新员工的工资编号与现存的工资编号相同时添加新员工）				
没有正在匹配/合并的输入文件或事务的匹配/合并（用以验证是否正确地关闭了匹配/合并文件）				
删除文件中第一个条目的匹配/合并				
删除附加/合并文件最后一个条目的匹配/合并				
两个收入记录具有相同值的匹配/合并				
两个收入记录的值是匹配/合并文件中将要删除的值的匹配/合并				
匹配/合并文件中删除的最后一个剩余记录				
收入匹配/合并文件超出序列或者存在某个单一记录超出序列的匹配/合并				
这些测试条件是否应用于可能在将要测试的软件中发生的匹配/合并条件总数				

511

512

F.13 压力测试检查表

下面的压力测试用于验证处理大量事务的软件的性能。

条 目	是	否	N/A	备 注
1. 是否确定了全部所需的性能?				
2. 是否确定了用于测试的所有系统特性?				
3. 是否确定了下列系统性能?				
数据输入操作符性能				
通信线路性能				
周转性能				
易用性及正常运行性能				
响应时间性能				
错误处理性能				
报告生成性能				
内部计算性能				
开发活动性能				
4. 是否确认了下列系统特性（可能会削弱性能）?				
内部计算器处理速度				
通信线路传输速度				
编程语言的效率				
数据库管理系统的效率				
输入终端及登录站的数目				
数据录入人员的能力水平				
备份计算机终端				
数据录入人员的候补				
中央处理站的期望停机时间				
异常软件终止的期望频率及持续时间				
排队能力				
文件存储能力				
5. 压力条件是否能够验证软件性能（由项目人员确定）?				

513

F.14 属性测试检查表

下面的属性测试包括验证正在测试的系统的质量及生产力特性的属性。示例中包括了软件引

入变更的容易程度。

条 目	是	否	N/A	备 注
1. 是否识别出了软件属性?				
2. 是否将软件属性分类?				
3. 最终用户或客户是否同意属性的分类?				
4. 是否为非常重要的属性开发好了测试条件?				
5. 对于正确性属性, 功能是否正确、完整?				
6. 对于完整性属性, 是否验证了每个文件或子模式的完整性?				
7. 对于认证属性, 是否对每个事务进行了认证处理?				
8. 对于审计追踪属性, 能否重建每个业务事务?				
9. 对于处理连续性属性, 能否在合理的时间内恢复系统, 并且在恢复期间捕获或处理事务?				
10. 对于服务属性, 周转及响应时间是否满足用户需求?				
11. 对于访问控制属性, 系统是否会限制认证用户?				
12. 兼容性属性是否遵循MIS标准、系统开发方法以及适当的政策、规程及规则?				
13. 对于可靠性属性, 能否正确处理错误、不完整或过期的数据?				
14. 对于易用性属性, 用户能否有效、经济地使用系统?				
15. 对于可维护性属性, 能否在合理投入的范围内变更或改善系统?				
16. 对于可移植属性, 能否将软件有效地移到其他平台?				
17. 对于耦合属性, 能否将软件正确地集成到其他系统?				
18. 对于性能属性, 最终用户能否接受软件性能?				
19. 对于易操作性属性, 操作人员能否有效、经济、高效地使用软件?				

514

515

F.15 状态测试检查表

下面的状态测试用于验证与可能发生的操作和功能环境相关的特殊条件。

项 目	是	否	N/A	备 注
1. 是否已经验证空主文件的状态?				
2. 是否已经验证空事务文件的状态?				
3. 是否已经验证丢失主记录的状态?				
4. 是否已经验证双主记录的状态?				
5. 是否已经验证空表的状态?				
6. 是否已经验证数量不足的状态?				

(续)

项 目	是	否	N/A	备 注
7. 是否已经验证负余额的状态?				
8. 是否已经验证重复输入的状态?				
9. 是否已经验证两次输入相同事务(尤其是从终端输入)的状态?				
10. 是否已经验证并发更新(例如,两个终端同时访问同一个主记录)的状态?				
11. 如下状态是否被验证:对服务和产品的要求比支持它们的服务和产品更高的情况?				

516

F.16 规程测试检查表

下面的规程测试用于验证软件的操作、终端及通信流程。

项 目	是	否	N/A	备 注
1. 是否已经验证启动规程?				
2. 是否已经验证查询规程?				
3. 是否已经验证文件挂载规程?				
4. 是否已经验证更新规程?				
5. 是否已经验证备份规程?				
6. 是否已经验证离线存储规程?				
7. 是否已经验证恢复规程?				
8. 是否已经验证终端操作规程?				
9. 当主计算机停机时操作终端所需要的规程是否已经得到验证?				
10. 当终端停机时捕获数据所需要的规程是否已经得到验证?				

517

F.17 控制测试检查表

下面的控制测试用于验证内部控制支持准确的、完整的、及时的以及授权的处理的能力。这些控制通常由评估控制充分性的审核者来验证。

条 目	是	否	N/A	备 注
1. 软件处理的业务事务是否已经识别出来?				
2. 是否为每个事务准备了事务流分析?				
3. 是否记录了事务流的控制?				
4. 数据输入控制处理了以下属性吗?				
数据输入的准确性				

(续)

条 目	是	否	N/A	备 注
数据输入的完整性				
数据输入的时间线				
将数据输入转换成机器可处理的格式				
输入按键				
数据输入处理进度				
数据输入职责的分配（创建、输入及处理数据，并分配输出）				
最终用户的输入（有控制组的帮助）				
所有源文件的输入				
批处理技术				
记录计数				
预确定控制总数				
控制日志				
按键验证				
预编程按键格式				
编辑输入				
输入数据元素				
数据确认编辑技术				
覆盖及旁路监控				
限制覆盖及旁路以监管人员				
为管理者自动记录并提交覆盖及旁路以进行分析				
在数据输入期间自动扩展控制数目				
记录事务错误				
为修正并重新录入事务监控不合格事务				
数据输入处理的写入规程				
关于所有数据错误状况的适当的出错消息				
数据输入终端的安全性				
通过终端输入的业务事务密码				
在预定义的不活动的时间之后关闭终端				
报告未授权终端的使用				
为终端内置验证代码				
通过终端输入的事务的日志，交互显示以告知终端操作者需要输入的数据				
5. 数据输入控制是否包含以下控制？				
新数据的精确度				

518

519

(续)

条 目	是	否	N/A	备 注
新数据的完整性				
及时记录新数据				
创建新数据的过程及方法				
空白源文件的安全性				
交叉引用域的检查				
预先编号的文档				
事务认证				
系统覆盖				
手动调整				
源文件的批处理				
源文件的控制总数				
源文件错误的修正流程				
源文件的记忆库				
数据输入的源文件的传输				
由数据输入功能确认的已输入文档的源文件功能(对于联机系统,同时执行数据创建与数据输入)				
数据输入的提示信息				
6. 流程控制处理以下内容吗?				
贯穿处理过程的输入				
如何控制处理过程的说明				
异常终止或异常状况				
管理者需要评审的操作日志				
核对记录数及控制总数的规程				
处理控制总数以及手动扩展控制总数的核对				
确保执行程序的正确版本的规程				
确保使用文件的正确版本的规程				
持续运行总数的维护				
从最后一个到当前运行(或者两个不同时期之间)的处理过程的调和				
验证新数据				
处理后手动验证覆盖及旁路规程				
事务历史文件的维护				
控制错误的规程				
修正并及时重新进入被拒绝的事务				

(续)

条 目	是	否	N/A	备 注
并发更新保护规程				
为每个错误状况显示出错消息				
修正规程及原始事务的同一规程				
7. 数据输出控制是否处理了以下内容?				
账目文档 (如银行支票)				
在输出准备中被破坏的账目文档				
输出的完整性				
为验证可接受性及完整性评审输出文档				
核对输出文档的记录数及控制总数				
输出产品的识别				
将输出产品提交至正确的场所				
输出产品及时交付				
输出产品的适当安全				
为所有输出的准确性而分配的最终用户的职责				
输出产品及交付物的日志				
清除输出出错消息?				
输出错误的历史信息				
告知用户输出产品错误				
告知用户异常终止				
在了解输出的过程中用户可以寻求帮助的电话号码				
在了解关于输出产品进度的信息时用户可以寻求帮助的电话号码				
输出副本的数目				
决定获取联机输出人员的规程				
联机输出的控制总数				
联机输出的写入规程				
建立在输出信息基础上的用户响应规程				
8. 是否确定了每个控制范围的风险级别?				
9. 是否确定了由审计部门确定的风险级别?				
10. 是否告知了最终用户或客户控制风险的级别?				

521

522

F.18 控制流程测试检查表

下面的控制流程测试确认了事务在被测系统中的控制流程。可以在处理过程中确定记录是否丢失或被错误处理。

523

条 目	是	否	N/A	备 注
1. 是否已经测试了两个方向的所有分支?				
2. 是否已经执行了所有语句?				
3. 是否已经测试了所有循环?				
4. 是否已经测试了每个循环的所有迭代?				
5. 是否已经测试了所有的执行路径?				
6. 是否已经调入并在测试过程中执行了所有的子程序及程序库?				

F.19 测试工具选择检查表

为项目寻找合适的工具是一件很困难的工作。在选择工具之前首先要回答几个问题。下面的测试工具选择检查表就列出了这样一些问题，帮助QA小组评价和选择自动化测试工具。

524

条 目	是	否	N/A	备 注
1. 测试人员使用这种工具的难易程度如何? 是能够快速地掌握使用方法, 还是需要培训?				
2. 是否有小组成员具备使用这种工具的经验?				
3. 如果需要培训, 是授课、书籍还是其他形式的教授方式?				
4. 这种工具能否在现有的计算机系统上有效地运行?				
5. 是否需要更多内存、更快的处理器等?				
6. 工具是否容易使用?				
7. 用户界面是否友好?				
8. 是否会导致用户错误?				
9. 这种工具是否能测试你的应用程序? 许多测试工具只能在图形用户界面环境下运行, 而其余的测试只能在非图形用户界面环境下运行。				
10. 这种工具能否处理整个项目测试? 也就是说, 如果在扩展时间周期内是否还能运行数百次(如果不是数千次的话)?				
11. 这种工具能否长时间运行而不崩溃, 还是工具本身就具有很多错误?				
12. 与现在或之前使用过这种工具的用户进行沟通。看这种工具是否满足了他们的需求?				
13. 他们的测试需求与你的有多少相似之处? 工具是否运行良好?				
14. 尝试去选择先进一些的工具, 这样日后升级时也会保持良好的性价比。				
15. 如果有演示版本, 在决定使用之前要试用一下。				
16. 这种工具的价格是否符合QA部门或公司的预算?				
17. 这种工具是否满足公司的测试方法的需求?				

F.20 项目信息收集检查表

这个检查表用于验证信息的易用性，并且需要在项目的初始进行。对于所有否定的回答，QA测试经理会评价其影响并当作问题记录下来以报告给各方解决。这项工作可以通过每周状态报告或电子邮件完成。

上下文	活 动	是	否	备 注
提议阶段				
	QA小组是否准备好进行QA评估?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否对提案进行了评审并获得批准?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否完成了估算及风险评估?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否将启动工作产品发送到项目服务器?	<input type="checkbox"/>	<input type="checkbox"/>	
	供应商合同(如果可行)	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已对合同进行了评审并获得批准?	<input type="checkbox"/>	<input type="checkbox"/>	需要在QA项目计划中定义的流程
	是否存在主合同及提案?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否定义了提案及联络方式?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否定义了项目验收文件/联络方式?	<input type="checkbox"/>	<input type="checkbox"/>	
项目开始				
	是否创建了项目文件夹?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否对项目经理进行了培训?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否完成了项目启动会议?	<input type="checkbox"/>	<input type="checkbox"/>	
	提出提案的经理是否参加了项目启动会议?	<input type="checkbox"/>	<input type="checkbox"/>	
项目计划及进度				
	是否将审计与评审列入了计划内?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否确定了项目目标?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否讨论过配置管理?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否讨论过人员计划?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否讨论过培训计划?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否讨论过状态报告方法及频率?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否讨论过项目进度?	<input type="checkbox"/>	<input type="checkbox"/>	
	项目进度是否包括了项目中的所有活动?	<input type="checkbox"/>	<input type="checkbox"/>	
	QA项目管理计划是否经过项目经理及其他人员的评审?	<input type="checkbox"/>	<input type="checkbox"/>	
	QA项目进度是否已经过团队评审?	<input type="checkbox"/>	<input type="checkbox"/>	
测试过程概述				
	项目经理是否评审和批准了测试过程?	<input type="checkbox"/>	<input type="checkbox"/>	

525

526

(续)

上下文	活 动	是	否	备 注
项目文件夹				
	是否讨论了预算及风险?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否讨论了角色及职责?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否对关键性资源进行了计划?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否确认了项目所依赖的资源?	<input type="checkbox"/>	<input type="checkbox"/>	
	通过完整性质量测试对项目文件夹进行了评审吗?	<input type="checkbox"/>	<input type="checkbox"/>	

F.21 影响分析检查表

影响分析检查表用于帮助分析变更对系统产生的影响。对于所有否定的回答,测试经理会评价其影响并当作问题记录下来以报告给相关各方解决。这项工作可以通过每周状态报告或电子邮件完成。

上下文	活 动	是	否	备 注
527	增强的业务需求是否可用?	<input type="checkbox"/>	<input type="checkbox"/>	
	新需求的新功能规约是否可用?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否理解了额外/新的需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	新版本的原型文档是否可用?	<input type="checkbox"/>	<input type="checkbox"/>	
	能否确认所提出的变更?	<input type="checkbox"/>	<input type="checkbox"/>	
	能否确认受到提高性能影响的应用程序?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为提高性能而精确定义了测试范围?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为提高性能及受到影响的应用区域准备了测试环境/用例?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否制定了测试计划/策略?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为所有的环境/用例准备了测试数据需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为新的/额外的需求完善了自动化范围?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否分析了对现存脚本的影响?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为新版本记录了测试执行计划?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否准备了可追踪矩阵文档?	<input type="checkbox"/>	<input type="checkbox"/>	
	新需求是否带来了架构变更?	<input type="checkbox"/>	<input type="checkbox"/>	
	新需求是否带来了数据库的变更?	<input type="checkbox"/>	<input type="checkbox"/>	
528	是否分析了由新需求带来的图形用户界面变更?	<input type="checkbox"/>	<input type="checkbox"/>	

F.22 环境准备检查表

环境准备检查表用于在开始测试执行前验证环境的就绪情况。对于所有否定的回答,测试经理将评价其影响并当作问题记录下来以报告给相关各方解决。这项工作可以通过每周状态报告或

电子邮件完成。

上下文	即将检查的条目	是	否	备 注
	客户是否签署了测试策略？	<input type="checkbox"/>	<input type="checkbox"/>	
	测试环境是否已准备妥当？	<input type="checkbox"/>	<input type="checkbox"/>	
	硬件	<input type="checkbox"/>	<input type="checkbox"/>	
	<输入每个组件>	<input type="checkbox"/>	<input type="checkbox"/>	
	软件	<input type="checkbox"/>	<input type="checkbox"/>	
	<输入每个组件>	<input type="checkbox"/>	<input type="checkbox"/>	
	是否创建了实验台？	<input type="checkbox"/>	<input type="checkbox"/>	
	每个期望格式的数据是否可用（测试数据方针——计划）？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否真实地迁移数据？	<input type="checkbox"/>	<input type="checkbox"/>	
	迁移数据的充分性？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否完成了软件传输及初始版本的加载（加载管理）？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否创建了用户ID和密码，用以从客户端/开发端访问环境？	<input type="checkbox"/>	<input type="checkbox"/>	
	测试环境搭建	<input type="checkbox"/>	<input type="checkbox"/>	
	测试小组是否已做好准备工作以开始测试？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否完成了测试实验室的搭建？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否如测试策略所记录的那样定义并建立了交互模式（项目计划）？	<input type="checkbox"/>	<input type="checkbox"/>	
	客户是否了解测试策略中定义的缺陷管理过程？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个项目策略计划定义并建立了进入标准？	<input type="checkbox"/>	<input type="checkbox"/>	
	<在此输入每个标准>	<input type="checkbox"/>	<input type="checkbox"/>	
其他潜在问题：				

529

F.23 项目完成情况检查表

项目完成情况检查表用于确认项目的所有关键活动都已完成。

上下文	活 动	状 态			备 注
		是	否	必须（R）/可选（O）	
	是否执行了所有测试用例？	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否关闭或延迟了所有缺陷？	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否关闭了所有变更请求？	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否验证了已提交的软件库？	<input type="checkbox"/>	<input type="checkbox"/>	O	
	是否完成了用户培训？	<input type="checkbox"/>	<input type="checkbox"/>	O	

(续)

上下文	活 动	状 态			备 注
		是	否	必须 (R) / 可选 (O)	
530	需要提交的文件是否已移交给客户?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否从客户处获得了项目签署?	<input type="checkbox"/>	<input type="checkbox"/>	O	
	项目目录是否包含文档的最新版本?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	所有文档是否已存档并输入至数据库?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否将客户反馈表发给客户?	<input type="checkbox"/>	<input type="checkbox"/>	O	
	客户提供的材料是否已经被返回或发布给其他的项目, 并且这些材料是否与客户进行过交流?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	在正式关闭项目时是否进行了沟通? (客户、高级主管、本土团队、质量团队、沟通组及项目团队)	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否备份了项目目录?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否将设备存放在防火仓内?	<input type="checkbox"/>	<input type="checkbox"/>	R	
531	是否将项目目录从服务器中隔离出来?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	项目数据库中的项目是否标记为已关闭?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否完成了度量数据采集?	<input type="checkbox"/>	<input type="checkbox"/>	R	
	是否更新了技能数据库?	<input type="checkbox"/>	<input type="checkbox"/>	O	

F.24 单元测试检查表

单元测试检查表用于验证单元测试执行的全面性。

期望的测试活动	已完成			备注/说明
	是	否	N/A	
是否验证了每个字段只有正确格式的数据才能输入[如数字(有符号的/无符号的)、字母、数字字母(特殊字符)、日期、有效及无效]? 检查无效数据的出错消息?				
是否验证了每个字段只有允许值才能输入(如表格、范围、最大值及最小值)? 检查无效数据的出错消息?				
是否验证了每个字段都需要遵守的业务规则(如存在其他字段时的强制/非强制内容, 关系编辑)?				
是否测试了每个出错消息?				
是否验证了每一个字段以处理无效数据?				
532 是否验证了所有大写及小写的字段条件?				
是否验证或处理了所有内部表的充分性以提供最大容量(如数据集的分布、接受事务的数量)? 是否检查了出错消息?				
对于数值字段, 是否测试了所有的“0”值?				

(续)

期望的测试活动	已 完 成			备注/说明
	是	否	N/A	
是否基于数据字典定义测试了所有无效数据?				
是否对需求规约进行了评审以保证测试了所有条件?				
是否验证了所有字母字段中的“空”条件?				
是否验证了数据都从正确的物理数据库读取与写入?				
是否正确初始化了所有的字段?				
是否验证了所有受保护的字段?				
是否验证了数据都从适当的文件中读取和写入并且验证了字段?				
是否验证了每个计算都能在基于业务规则的全部相关数据项的范围内生成正确的结果?				
是否每个输出及其格式都已被验证(如进位与截断)?				
对传送给其他系统的数据是否是接收系统需要的正确格式进行了验证?				
其他系统传送的数据格式是否正确?				
是否对所有在设计规约中明确的安全方面的需求进行了验证?				
是否所有输出都被验证其安全级别的类型与提示信息相吻合?				
是否按照软件正确运行环境标准捕获与处理了所有的错误条件(除错误代码与出错消息)?				
是否验证了软件在发生一些意外错误时没有留下了垃圾数据(如一般保护错误、语法错误、异常退出等)?				
是否所有的提示信息都由软件的典型最终用户验证过是清楚并且容易理解的?				
说明书的典型用户是否验证了所有的使用说明书简明扼要并且容易理解?				
文档的典型用户是否验证了文档是清楚并且是容易理解的?				
根据REL IT标准, 是否所有的Tab键、按钮、超链接以及Tab的焦点顺序符合逻辑?				
是否验证了所有的命令在使用鼠标和键盘的情况下都可用?				
是否完成了响应时间满足需求的测试? 在软件执行环境(大数据量的运行)中响应时间是否是可接受的?				
是否评审了代码?				
是否验证了所有未定义的循环迭代?				
是否满足了所有的编程标准?				
是否验证了无效代码?				
是否验证了无效的数据关系?				
是否验证了无效的数据格式?				
客户是否签署了测试策略?				

533

534

(续)

期望的测试活动	已 完 成			备注/说明
	是	否	N/A	
是否对软件条目满足所有标准进行了验证,用以适应期望执行软件条目的环境?				
是否验证了软件条目满足所有财务控制与隐私的共同标准的强制要求?				
软件能否被需要执行它的具体环境所接受?				
备注:				
完成者: _____ 日期: _____ 开发者: _____	验收者: _____ 日期: _____ 开发经理: _____			

F.25 歧义性评审检查表

歧义性评审检查表用于辅助为了发现结构歧义性而对功能规约进行的评审(不要与内容评审混淆)。对于所有否定的回答,QA项目经理会评价其影响并当作问题记录下来以报告给相关各方解决。这项工作可以通过每周状态报告或电子邮件完成。

上 下 文	任 务	是	否	备 注
复杂性	需求是否过于复杂?	<input type="checkbox"/>	<input type="checkbox"/>	
其他不确定	丢失了条件中的其他部分?	<input type="checkbox"/>	<input type="checkbox"/>	
参考内容有歧义	没有明确定义参考内容?	<input type="checkbox"/>	<input type="checkbox"/>	
活动范围	没有明确定义活动范围?	<input type="checkbox"/>	<input type="checkbox"/>	
冗余	只有原因没有结果?	<input type="checkbox"/>	<input type="checkbox"/>	
	缺少结果?	<input type="checkbox"/>	<input type="checkbox"/>	
	只有结果没有原因?	<input type="checkbox"/>	<input type="checkbox"/>	
	缺少原因?	<input type="checkbox"/>	<input type="checkbox"/>	
有歧义的逻辑操作符	是否存在不清晰的and/or的组合使用?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在隐含的连接符?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否正确使用了or?	<input type="checkbox"/>	<input type="checkbox"/>	
否定	是否存在范围否定?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在不必要的否定?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在双重否定?	<input type="checkbox"/>	<input type="checkbox"/>	
有歧义的语句	是否存在有歧义的动词?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在有歧义的副词?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在有歧义的形容词?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在有歧义的变量?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在别名?	<input type="checkbox"/>	<input type="checkbox"/>	

(续)

上 下 文	任 务	是	否	备 注
随机组织	是否存在混合的原因与结果?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在随机用例序列?	<input type="checkbox"/>	<input type="checkbox"/>	
内置的歧义性	是否具备功能/环境知识?	<input type="checkbox"/>	<input type="checkbox"/>	
不清楚的优先关系	是否存在不明确的事件序列?	<input type="checkbox"/>	<input type="checkbox"/>	
隐含用例	是否存在隐含的用例?	<input type="checkbox"/>	<input type="checkbox"/>	
等等	是否存在“等等”的例子?	<input type="checkbox"/>	<input type="checkbox"/>	
“即”(i.e.)与“如”(e.g.)	是否正确使用了“即”与“如”?	<input type="checkbox"/>	<input type="checkbox"/>	
时间歧义性	是否存在时间歧义性?	<input type="checkbox"/>	<input type="checkbox"/>	
边界歧义性	是否存在边界歧义性?	<input type="checkbox"/>	<input type="checkbox"/>	

537

F.26 架构评审检查表

架构评审检查表用于评审架构的完整性和清晰性。对于所有否定的回答,测试经理会评价其影响并当作问题记录下来以报告给相关各方解决。这项工作可以通过每周状态报告或电子邮件完成。

上 下 文	任 务	是	否	备 注
	是否记录了系统的一个整体描述?			
	是否定义了2层或3层的架构?			
	是否定义了数据库和存取?			
	是否定义了服务器?			
	是否定义了协议,如HTTP、JSP、PeopleSoft、Tuxedo等?			
	是内部供应商还是外部供应商?			
	是否定义了解决相关技术架构问题的联系点?			
	是否定义了平台?			
	是否有一个网络图?			
	是否定义了测试设备?			
	是否定义了负载平衡?			
	是否定义了业务处理?			
	是否某些任务的执行比另外一些更普遍?			
	是否定义了峰值?			
	是否定义了Web服务器?			

538

F.27 数据设计评审检查表

数据设计评审检查表用于评审逻辑设计及物理设计内容的完整性和清晰性。对于所有否定的回答,QA项目经理将评价其影响并当作问题记录下来以报告给相关各方解决。这项工作可以通

过每周状态报告或电子邮件完成。

上 下 文	任 务	状 态		
		是	否	备 注
逻辑设计				
	存在定义不充分的数据？	<input type="checkbox"/>	<input type="checkbox"/>	
	数据实体定义不完全？	<input type="checkbox"/>	<input type="checkbox"/>	
	维度定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	属性定义充分吗？	<input type="checkbox"/>	<input type="checkbox"/>	
	有违反规范化的情况吗？	<input type="checkbox"/>	<input type="checkbox"/>	
	主键定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	外键定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	混合键定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	实体子类型定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	父进程不完整？	<input type="checkbox"/>	<input type="checkbox"/>	
	子进程不完整？	<input type="checkbox"/>	<input type="checkbox"/>	
	进程输入、输出与实体的交互不完全？	<input type="checkbox"/>	<input type="checkbox"/>	
	基本的实体定义正确吗？	<input type="checkbox"/>	<input type="checkbox"/>	
	有并行链接问题吗？	<input type="checkbox"/>	<input type="checkbox"/>	
	事件触发进程设计不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	实体/进程的关联定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	实体/进程的读关联定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	实体/进程的更新关联定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	
	实体/进程的删除关联定义不正确？	<input type="checkbox"/>	<input type="checkbox"/>	

F.28 功能规约评审检查表

功能规约评审检查表用于评审功能规约内容的完整性和清晰性（不要与歧义性评审混淆）。对于所有否定的回答，QA项目经理会评价其影响并当作问题记录下来以报告给相关各方解决。这项工作可以通过每周状态报告或电子邮件完成。

上 下 文	任 务	是	否	备 注
引言				
	是否文为功能规约的目标、范围和组织建立文档?	<input type="checkbox"/>	<input type="checkbox"/>	
软件概述				
产品描述	是否描述了产品开发的原因和一系列重要的特征和能力?	<input type="checkbox"/>	<input type="checkbox"/>	
产品功能	是否有一系列需要软件完成的功能?	<input type="checkbox"/>	<input type="checkbox"/>	
	对于大部分的功能，是否有一个表（或者一些其他格式）可以描述这些功能之间的关系？注意：这可以是对需求文档的更新。	<input type="checkbox"/>	<input type="checkbox"/>	

(续)

上 下 文	任 务	是	否	备 注
用户特征	是否描述了软件的目标用户的工作能力、特定的知识或技能的水平?	<input type="checkbox"/>	<input type="checkbox"/>	
用户操作和实践	是否描述了用户通常如何使用软件和执行任务?	<input type="checkbox"/>	<input type="checkbox"/>	
通用限制	是否描述了算法限制、用户界面限制和数据限制?	<input type="checkbox"/>	<input type="checkbox"/>	
假设	是否描述了所有的假设?	<input type="checkbox"/>	<input type="checkbox"/>	
其他的软件	是否描述了系统如何与其他软件系统交互?	<input type="checkbox"/>	<input type="checkbox"/>	
具体功能描述				
描述	是否每一个功能的职责都有所描述?	<input type="checkbox"/>	<input type="checkbox"/>	
输入	是否指定了所有的输入来源?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有输入的精确需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的输入范围值?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了输入的频率?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有输入的格式?	<input type="checkbox"/>	<input type="checkbox"/>	
处理	计算如果使用了特定的使用方法和标准,它们是否可引用?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否定义了数据库?	<input type="checkbox"/>	<input type="checkbox"/>	
输出	是否描述了功能的输出?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否存在一个用户界面?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的输出目的?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有输出的精确需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的输出范围值?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有输出的频率?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有输出的格式?	<input type="checkbox"/>	<input type="checkbox"/>	
报告				
	是否指定了所有报告的格式?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有报告中所使用的计算方法/公式?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有报告的数据过滤需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有报告排序需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了报告总汇需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了报告格式化需求?	<input type="checkbox"/>	<input type="checkbox"/>	
非功能性				
	是否为每个功能指定了所有的性能需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个功能指定了所有的设计约束?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个功能指定了所有的属性?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个功能指定了所有的安全性需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个功能指定了所有的可维护性需求?	<input type="checkbox"/>	<input type="checkbox"/>	

541

542

(续)

上 下 文	任 务	是	否	备 注
	是否为每个功能指定了所有的数据库需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个功能指定了所有的可操作需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个功能指定了所有的安装需求?	<input type="checkbox"/>	<input type="checkbox"/>	
接口				
	是否指定了所有的用户接口?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的批处理接口?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的硬件接口?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的软件接口?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的沟通接口?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的接口设计约束?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的接口的安全性需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有接口的可维护性需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否为每个用户界面指定了所有的人机交互界面?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否确定了所有的内部接口?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的内部功能接口特征?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否描述了所有的出错消息需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否描述了输入范围检查需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否定义了与用户喜好相对应的选择顺序和屏幕?	<input type="checkbox"/>	<input type="checkbox"/>	
附加需求				
数据库	是否定义了所有与数据库相关的需求(如数据库的类型、处理大的文本字段的能力、实时的能力、多用户的能力)以及与查询、表单相关的具体需求?	<input type="checkbox"/>	<input type="checkbox"/>	
管理	是否定义了定期更新和数据管理需求?	<input type="checkbox"/>	<input type="checkbox"/>	
用户文档	是否定义了与软件一起交付的用户文档需求?	<input type="checkbox"/>	<input type="checkbox"/>	
其他需求	是否还存在上文中没有提到的软件设计过程中需要考虑的需求?	<input type="checkbox"/>	<input type="checkbox"/>	
时间				
	是否指定了所有期望的处理时间?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的数据转化率?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有的系统吞吐率?	<input type="checkbox"/>	<input type="checkbox"/>	
硬件	是否指定了最小内存量?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了最小存储量?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了最大内存量?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了最大存储量?	<input type="checkbox"/>	<input type="checkbox"/>	
软件	是否指定了所需的软件环境/操作系统?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所有所需的软件工具?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了需要购买的所有用户系统所需使用的软件?	<input type="checkbox"/>	<input type="checkbox"/>	

(续)

上 下 文	任 务	是	否	备 注
网络	是否指定了目标网络?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所需的网络协议?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所需的网络容量?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了所需的/估算的网络吞吐率?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了估算的网络连接数?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否定义了最低网络性能需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否定义了最高网络性能需求?	<input type="checkbox"/>	<input type="checkbox"/>	

545

F.29 原型评审检查表

原型评审检查表用于评审原型的完整性和清晰性。对于所有否定的回答，测试经理会评价其影响并当作问题记录下来以报告给相关各方处理。这项工作可以通过每周状态报告或电子邮件完成。

上 下 文	条 目	是	否	备 注
	原型是否反映了最初的客户需求?			
	原型设计是否反映了最初的需求?			
	是否创建了详细的交互/可视用户界面?			
	是否能很容易地将用户界面组件和底层的功能行为连接起来?			
	原型工具是否提供了语言学习的捷径?			
	针对所得的原型工具语言的修改是否能够很容易地完成?			
	简单性：是否如最初的需求中所述，用户界面提供了一个合适的方法允许用户去访问底层的功能行为?			
	原型使用很简单吗?			
	简明性：原型是否包含了完景的用户界面而没有无关的详细内容?			
	原型是否包含了一个数据模型，可以为应用程序本身定义数据结构?			
	是否能够代表瞬时/持久数据?			
	原型是否需要新的需求?			
	原型是否对定义的需求处理不当?			

546

F.30 需求评审检查表

需求评审检查表用于验证测试项目需求的全面性和完整性。对于所有否定的回答，测试经理会评价其影响并当作问题记录下来以报告给相关各方处理。这项工作可以通过每周状态报告或电子邮件完成。

上 下 文	任 务	状 态		备 注
		是	否	
清晰性				
	需求是否是用非技术性语言编写的？	<input type="checkbox"/>	<input type="checkbox"/>	
547	最终产品的每一个特征是否都是使用统一的术语描述的？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否使用术语表来统一定义术语的特定含义？	<input type="checkbox"/>	<input type="checkbox"/>	
	其他独立的团队是否可以理解和执行需求？	<input type="checkbox"/>	<input type="checkbox"/>	
完整性				
	是否存在内容的索引表？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否对所有的图表进行了标号？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有图表都是交叉索引的？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有的需求都已经定义？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与功能内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与性能内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与设计限制内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与属性内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与外部界面内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与数据库内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与软件内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与硬件内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与输入内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
548	是否包括了所有与输出内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与报告内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与安全性内容相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与可维护性相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否包括了所有与危机性相关的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了对需求规约可能做的变更？	<input type="checkbox"/>	<input type="checkbox"/>	
一致性				
	是否有描述同样的事物的需求与代表某些术语的其他需求冲突？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有描述同样的事物的需求与代表属性的其他需求冲突？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有需求描述了两个或两个以上在逻辑上有冲突的行为？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有需求描述了两个或者两个以上在时间上有冲突的行为？	<input type="checkbox"/>	<input type="checkbox"/>	
可追溯性				
	是否所有的需求可以追溯到某一个特定的用户需求？	<input type="checkbox"/>	<input type="checkbox"/>	

(续)

上 下 文	任 务	状 态		备 注
		是	否	
	是否所有的需求可以追溯到一个特定的源文档或者人?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有的需求可以追溯到特定的设计文档?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有的需求可以追溯到特定的软件模块?	<input type="checkbox"/>	<input type="checkbox"/>	
可验证性				
	是否包括了不可能实现的需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	对于每一个需求, 是否都存在一个人或者一台机器可以执行的过程可以验证需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有需求是用以后才可以验证的术语表达的?	<input type="checkbox"/>	<input type="checkbox"/>	
可修改性				
	需求文档的组织是否清晰、有逻辑性?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否采用已接受的标准来组织?	<input type="checkbox"/>	<input type="checkbox"/>	
内容				
通用				
	是否每一个需求都与问题及其解决方式相关?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否真正详细地设计了定义的需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否真正详细地验证了定义的需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否真正通过项目管理的方式来管理需求?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有引言部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有总体描述部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有范围部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有定义、首字母缩写、简写部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有产品概述部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有产品功能部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有用户特征部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有通用限制条件部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有假设和依赖部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否有具体的需求部分?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有的必要的附录都存在?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有必要的数字都存在?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有必要的表都存在?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否所有必要的图存在?	<input type="checkbox"/>	<input type="checkbox"/>	
可靠性				
	是否为每个需求定义了特定的软件失效的后果?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了防止失效的信息?	<input type="checkbox"/>	<input type="checkbox"/>	

549

550

(续)

上 下 文	任 务	状 态		备 注
		是	否	
	是否指定了错误检查策略?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否指定了错误纠正策略?	<input type="checkbox"/>	<input type="checkbox"/>	
硬件				
	是否指定了具体的硬件细节?	<input type="checkbox"/>	<input type="checkbox"/>	
软件				
	是否指定了具体的软件细节?	<input type="checkbox"/>	<input type="checkbox"/>	
通信				
	是否指定了所有的通信和网络细节?	<input type="checkbox"/>	<input type="checkbox"/>	

551

F.31 技术设计评审检查表

技术设计评审检查表用于评审技术设计的清晰性和完整性。对于所有否定的回答,QA项目经理会评价其影响并当作问题记录下来以报告给各方处理。这项工作可以依据严重程序通过每周状态报告或电子邮件完成。

上 下 文	任 务	是	否	备 注
技术设计				
	逻辑顺序错误?	<input type="checkbox"/>	<input type="checkbox"/>	
	处理不准确?	<input type="checkbox"/>	<input type="checkbox"/>	
	程序正确地处理输入或输出参数?	<input type="checkbox"/>	<input type="checkbox"/>	
	过程在许可范围内接受所有数据?	<input type="checkbox"/>	<input type="checkbox"/>	
	对输入数据进行了限制及有效性检查?	<input type="checkbox"/>	<input type="checkbox"/>	
	恢复程序没有实现或不充分?	<input type="checkbox"/>	<input type="checkbox"/>	
	所需逻辑缺失或不充分?	<input type="checkbox"/>	<input type="checkbox"/>	
	值错误或有歧义?	<input type="checkbox"/>	<input type="checkbox"/>	
	数据存储错误或不充分?	<input type="checkbox"/>	<input type="checkbox"/>	
	变量缺失或者没有正确声明?	<input type="checkbox"/>	<input type="checkbox"/>	
	数据库与数据环境不兼容?	<input type="checkbox"/>	<input type="checkbox"/>	
	模块结构表现为高内聚?	<input type="checkbox"/>	<input type="checkbox"/>	
	没有对算法的精确度和速度进行评估?	<input type="checkbox"/>	<input type="checkbox"/>	
	控制结构不可扩展?	<input type="checkbox"/>	<input type="checkbox"/>	
	控制结构忽略了处理过程的优先级?	<input type="checkbox"/>	<input type="checkbox"/>	
	接口协议使用不正确?	<input type="checkbox"/>	<input type="checkbox"/>	
	没有将数据转换为正确格式?	<input type="checkbox"/>	<input type="checkbox"/>	

552

(续)

上下文	任 务	是	否	备 注
	没有考虑舍入和截断带来的影响?	<input type="checkbox"/>	<input type="checkbox"/>	
	错误地使用了索引?	<input type="checkbox"/>	<input type="checkbox"/>	
	存在死循环?	<input type="checkbox"/>	<input type="checkbox"/>	
	违反了数据库规则?	<input type="checkbox"/>	<input type="checkbox"/>	
	没有覆盖特殊情况?	<input type="checkbox"/>	<input type="checkbox"/>	
	缺乏错误处理?	<input type="checkbox"/>	<input type="checkbox"/>	
	忽略了对时间的考虑?	<input type="checkbox"/>	<input type="checkbox"/>	
	接口规约被错误地理解或实现?	<input type="checkbox"/>	<input type="checkbox"/>	
	功能规约在各个软件模块中分配不当?	<input type="checkbox"/>	<input type="checkbox"/>	
	系统在功能方面达标, 但性能没有满足要求?	<input type="checkbox"/>	<input type="checkbox"/>	
	系统的复杂程度不够, 无法解决存在的问题?	<input type="checkbox"/>	<input type="checkbox"/>	
	对给定输入值的响应活动有不适当的或缺失?	<input type="checkbox"/>	<input type="checkbox"/>	
	输入某些特定值时, 算法近似值提供的精确度不够或者导致错误结果?	<input type="checkbox"/>	<input type="checkbox"/>	
	为解决特殊问题设计的详细的逻辑有错误?	<input type="checkbox"/>	<input type="checkbox"/>	
	异常或临界值的输入会导致意想不到的结果, 在代码内部无法正确解决?	<input type="checkbox"/>	<input type="checkbox"/>	
	算法没有覆盖到所有必需的情况?	<input type="checkbox"/>	<input type="checkbox"/>	
	存在错误的或者产生了错误解决方案的算法?	<input type="checkbox"/>	<input type="checkbox"/>	

553

F.32 测试用例准备工作评审检查表

使用这个检查表可以确保为每个需求规约都准备了测试用例。对于测试用例准备工作评审检查表中的所有否定的回答, 测试经理会评价其影响并记录下来以报告给各方处理。这项工作可以通过每周状态报告或电子邮件完成。

上下文	活 动	状 态		
		是	否	备 注
	已批准的测试计划是否可行?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已经准备好实现测试计划的资源?	<input type="checkbox"/>	<input type="checkbox"/>	
	基线文档是否可用?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已将领域知识传授给团队成员以便于处理应用程序?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已完成测试条件文档?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已为所有的需求设计了测试用例?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已验证可追溯性?	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已覆盖用例中的所有基本流?	<input type="checkbox"/>	<input type="checkbox"/>	

554

(续)

上 下 文	活 动	状 态		
		是	否	备 注
	是否已覆盖用例中的所有替换流？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已充分覆盖所有变更过的需求？	<input type="checkbox"/>	<input type="checkbox"/>	
	不可测试的需求是否已上升？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已为数据流穿越接口编写了测试用例？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已为项目计划中定义的所有类型的测试编写了测试用例？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已确定所有输出正确结果和错误结果的用例？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已确定所有的边界情况？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已为非功能需求编写了测试用例？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已为Web应用程序中的图形用户界面或超链接测试编写了测试用例？	<input type="checkbox"/>	<input type="checkbox"/>	
	是否已为测试数据完整性编写了测试用例？	<input type="checkbox"/>	<input type="checkbox"/>	

G.1 基本路径测试

基本路径测试是一种白盒测试技术，它根据程序执行时实际使用的流或者逻辑路径来定义测试用例。一条基本路径是程序在不迭代的情况下经过的一条唯一路径。基本路径是原子层次上的路径，系统的所有可能执行路径都是这些基本路径的线性组合。基本路径测试同时提供一个圈度量，通过检查程序控制结构来估量一个模块的源代码的复杂性。

为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个字段的取值范围。

程序：FIELD-COUNT

```
Node Statement
1.   Dowhile not EOF
      read record
2.       if FIELD_COUNTER > 7 then
3.   increment COUNTER_7 by 1
      else
4.       if FIELD_COUNTER > 3 then
5.   increment COUNTER_3 by 1
      else
6.   increment COUNTER_1 by 1
7.   endif
8.   endif
9.   End_While
10.  End
```

557

理论上来说，如果这个循环迭代了100次，那么则需要 1.5×10 个测试用例才能进行穷举测试，这是不可能完成的。而使用基本路径测试方法，程序只需要测试如下4条基本路径的测试用例：

```
1→10
1→2→3→8→9→1→10
1→2→4→5→7→8→9→1→10
1→2→4→6→7→8→9→1→10
```

数学上来讲，该程序的所有可能路径都可以由这4条基本路径的线性组合生成。经验告诉我

们，大部分潜在的缺陷都可以通过执行这4条基本路径的测试用例来发现，这也向我们展示了这项技术的能力。基本路径的数量同时也是循环复杂性的度量条件。一般我们建议程序模块的循环次数不要超过10。因为这种计算是一个很重的体力活，我们可以利用测试工具来将这个过程的自动化。想获得更详细的说明，请参见第六部分。

基本路径测试同样可以用于把程序模块集成起来的集成测试。这项技术的使用可以量化集成的相关工作量和设计层次上的复杂性。

G.2 黑盒测试

黑盒测试（又称功能测试）是一种测试条件以程序或系统的功能为基础生成的测试。也就是说，测试人员需要的信息是输入的数据和观察到的输出结果，但他们不知道程序或系统是怎样工作的。就像一个人不必知道汽车的内部是如何工作的就可以开车一样，我们不必知道程序的内部结构就可以执行它。测试人员侧重于根据规约去测试程序的功能。在黑盒测试中，测试人员把程序看做一个黑匣子，对程序或系统的内部结构并不关心。这一类测试包括决策表、等价类划分、范围测试、边界值测试、数据库完整性测试、因果图、正交表测试、数组和表测试、异常测试、极限测试、随机测试等。

558

黑盒测试的一个主要优点是测试活动本身的行为要与程序或系统的设计行为相吻合，这非常自然，所有人都可以理解。一些测试技术，如结构化走查、审查和JAD，可以证实这一点。黑盒测试的局限性是测试全部的、无遗漏的输入流是不太可能的，因为这要求每一个可能的输入条件或其组合都要被测试到。另外，因为测试人员不知道内部结构或处理逻辑，在黑盒测试中没有被测试到的部分很可能会有致命的错误或程序员故意放置一段代码而搞的恶作剧。例如，假设一个为公司开发工资管理程序的程序员在自己开发的代码中嵌入一段“保护”自己工作的代码，如果这位员工被辞退，他的员工代码在系统中不再存在，而他预置在程序里的判断迟早会发挥作用。

附加的程序逻辑：

```
if my employee ID exists
    deposit regular pay check into my bank account
else
    deposit an enormous amount of money into my bank account
    erase any possible financial audit trails
    erase this code
```

G.3 自底向上测试

自底向上测试技术是一种递增式的测试方法，这种技术首先对最底层的模块和系统组件进行集成并测试。紧接着，测试分层次的递增执行，一直到最顶层。经常需要驱动程序或者能调用当前测试模块或系统组件的临时测试程序。自底向上测试首先通过能调用相关接口的驱动程序来测试最底层的模块或系统组件。在这些基本组件被测试之后，程序或系统组件层次结构的下一个逻辑层次开始测试并不断驱动向更高的层次。

自底向上测试是大而复杂的系统测试的常用技术，它要经过相当长的一段时间才能使系统可

视化。软件的菜单和外部用户界面是最后被测试的，用户不能及早对这些界面和功能进行检查。自底向上测试技术有一个潜在的缺陷，就是需要花大量的时间来创建驱动程序，这可能会增加额外的错误。

G.4 边界值测试

边界值测试技术是一种黑盒测试技术，重点关注输入和输出等价类（详见等价类划分测试）的边界值。程序错误有在边界值附近集中的趋势。重点测试这些区域增加了发现程序错误的可能性。

559

边界值测试是等价类划分技术的一个变种，专注于每个等价类的边界值，例如，等于、大于和小于每个等价类。与在等价类中随机选取一个测试点不同，边界值分析采用一到多个测试用例来测试一个边界。焦点在输入空间（输入等价类）和输出空间（输出等价类）上。定义输出等价类相对比较困难，因此测试边界值。

由于输入和输出有很多变化情况，边界值测试需要创建大量的测试用例。我们推荐为每个输入的变量创建至少9个测试用例。输入的意义需要被透彻的理解，而且执行的行为必须和等价类相一致。边界值测试技术的一个限制条件是当包含复杂的计算时，确定等价类的范围变得十分困难。因此，这项技术需要系统需求被描述的尽可能的详细。下面是如何应用这项技术的一些例子。

1. 数值输入数据

范围区域。例如，“输入范围是整数0~100”，测试用例包括-1、0、100、101。

例如，“输入范围是实数0~100.0”，测试用例包括-0.000 01、0.0、100.0、100.000 01。

2. 数值输出数据

值的输出范围。例如，“保险统计表的输出数字范围是0.0~100 000.00美元”，应当构建输入条件使得输出结果分别为负数、0.0、100 000.00、100 000.01。

3. 非数值输入数据

表格或数组。例如，重点检查第一行和最后一行，比如对表格或数组进行读取、更新、写入、删除操作。

例如，尝试访问一个并不存在的表格或数组。

条目的数量。例如，“与一个型号相关联的产品数目最多为10个”；比如可以输入0个、10个和11个条目。

560

4. 非数字输出数据

表格或数组。例如，重点检查第一行和最后一行，比如对表格或数组进行更新、删除、插入操作。

输出的数量。例如，“最多同时显示10个股票”，比如可以尝试显示0个、10个和11个股票。

5. 图形用户界面

(1) 竖直和水平滚动到滚动条的两端。

(2) 颜色选择的上限和下限。

(3) 音量选择的上限和下限。

(4) 新的边界，比如一组可用输入值的可用边界值组合。

- (5) 微调控制项，比如在一个很小的编辑区域中放置两个半高的按钮。
- (6) 触发式的菜单项。
- (7) 列表框的边界。

G.5 分支覆盖测试

分支覆盖或者说判断覆盖是一种白盒测试技术，它的测试用例必须保证每个判断的真和假两种结果至少各出现过一次；举例来说，每个分支都必须至少被遍历一次。分支覆盖通常都满足语句覆盖（详见G31节），因为每条语句都在两条分支语句产生的子路径上。

为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个成员的取值范围。

程序：FIELD-COUNT

```

Dowhile not EOF
  read record
  if FIELD_COUNTER > 7 then
    increment COUNTER_7 by 1
  else
    if FIELD_COUNTER > 3 then
      increment COUNTER_3 by 1
    else
      increment COUNTER_1 by 1
    endif
  endif
endif
End_While
End

```

满足分支覆盖的测试用例如下所示：

测试用例	值 (FIELD_COUNTER)
1	>7, 比如8
2	≤7, 比如7
3	>3, 比如4
4	≤3, 比如3

对这个特定的例子，测试用例2是冗余的，可以删除。

G.6 分支/条件覆盖测试

分支/条件覆盖是一种白盒测试技术，它的测试用例要保证每个判断以及判断中的每个条件的所有可能的值都要被至少取到过一次。这是一种比判断或状态覆盖更强的逻辑覆盖技术，因为它覆盖了所有条件的组合，有些可能在单独的判断覆盖中不可能被测试到。条件覆盖同样满足语句覆盖。

使用这种技术创建测试用例的一种方法是建立一张真值表，将所有的条件和它们对应的反填入表中。接着，如果存在冗余的测试用例则应当删除。为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个成员的取值范围。

程序：FIELD-COUNT

```

Dowhile not EOF
  read record
  if FIELD_COUNTER > 7 then
    increment COUNTER_7 by 1
else
  if FIELD_COUNTER > 3 then
    increment COUNTER_3 by 1
else
  increment COUNTER_1 by 1
endif
endif
End_While
End

```

562

满足分支/条件覆盖的测试用例如下所示：

测试用例	值(FIELD_COUNTER)
1	>7, 比如8
2	≤7, 比如7
3	>3, 比如4
4	≤3, 比如3

在这个特殊的例子中，每个判断只有一个条件。如果有更多条件的话，每个条件和它的互补条件都应该被测试。同样，测试用例2是冗余的，可以删除。

G.7 因果图

因果图（也称石川图或鱼骨图）是分析令人不满意的情况的原因的非常有用的工具。这种方法具有几大优点，其中之一是这种方法给出了各个原因之间的关系的模拟表示，在初始搜索中，这被证明是一种有效的仿真方法。另一个优点是，通过提问为什么、什么、哪里、谁和怎样能够给出一种不断搜寻根本原因的方法。这是一种图表表示法，在这种表示法中，原因关系是很容易辨别的。

下面是因果图的一个应用示例，用于理解审查的过程。它发现：(1)有过多需要审查的材料会导致过多的准备工作；(2)过多的准备工作会导致过多的审查工作；(3)过多的审查工作反而会导致发现的缺陷更少。这一使用因果图进行的分析，通过限制要审查的材料量和准备工作的量，给出了优化审查过程的方法。

为建造因果图做适当的准备是必须的，可视性是一个关键的需求。在列出原因时，建议在

它们之间多留出一些空白，以便在后面的工作中给注释留出空间。

563 首先我们应当进行一些建造图表的阶段，从而得到最终完成的产品。这个步骤中，我们需要将因果图的小部分进行扩大，取出一个重要的原因并且使它成为另外一个因果图上要分析的结果。

因果图也可用于测试用例的设计，尤其在功能测试中。它们可以用于系统地选择一组测试用例，一组有很大可能性检测出系统错误的测试用例。这种技术研究了一个程序的输入和输入条件的混合，以开发测试用例，但是并不关心程序的内部行为。对于每一个产生的测试用例，这种技术还给出了预期的输出。输入和输出在需求规约中已经被定义好了（参见第六部分）。

下面是对于使用因果图将需求转换为测试用例的简要概述，后面附上一个如何应用此方法的例子。

1. 因果图法

- (1) 标识出所有需求。
- (2) 对需求加以分析并标出所有的原因和结果。
- (3) 给每个原因和结果分配一个唯一的编号。
- (4) 对需求加以分析并把它们表示为因果之间的布尔图。
- (5) 把图转换成决策表。
- (6) 将决策表中的列转换成测试用例。

例如，一个数据库管理系统需要数据库中的每一个文件的名字都在主索引中，主索引标识出每个文件的位置，索引被分割为10个部分。一个小系统正在开发中，这个系统允许用户互动地输入一条命令以在终端上显示索引的任意一个部分。因果图被用于为这个系统开发一组测试用例。在接下来的段落中，解释了系统的规约。

2. 规约

为了显示10个可能的索引部分中的一个，输入的命令必须由一个字母和一个数字组成，第一个输入的字符必须是D（显示）或者L（列出），而且这个字符必须在第一列中。第二个输入的字符必须是一个数字（0~9）在第二列中。如果遇到了这个命令，那么这个数字所标识的索引项将在终端上显示。如果第一个字符不正确，则显示“无效命令”；如果第二个字符不正确，则显示“无效索引编号”。

564 原因和结果按照如下方法标识。

原因

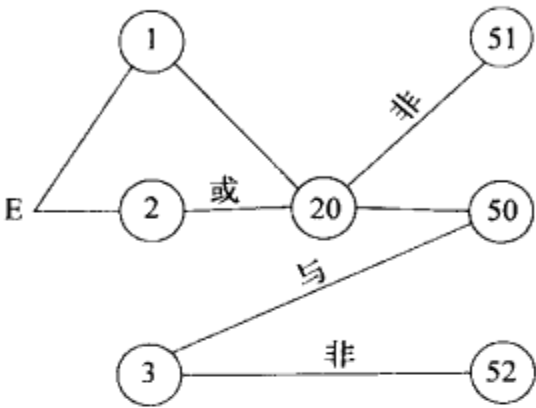
- (1) 第一列的字符是“D”。
- (2) 第一列的字符是“L”。
- (3) 第二列的字符是一个数字。

结果

- (1) 显示了索引项。
- (2) 显示了出错消息：“无效命令”。
- (3) 显示了出错消息：“无效索引编号”。

在对规约的分析过程中，我们建造了一个因果图（见图G-1），这是通过以下步骤完成的：

(1)将每个原因和结果用一个写有其唯一编号的节点表示;(2)将所有的原因节点竖直地列在图的左侧,将结果节点列在右侧;(3)通过分析规约,将原因和结果节点互相连接,每一个原因和结果可以处在两个状态之一,即真或者假,使用布尔逻辑设定原因的可能状态并且确定在何种条件下会给出那一种结果;(4)使用描述由于语法或者环境限制而导致的可能的因果组合的限制条件给图标加注释。



图G-1 因果图

节点20是一个代表节点1或者节点2的布尔状态的中间节点。如果节点20和节点3的状态都为真,那么节点50的状态也为真;如果节点1或者节点2为真,那么节点20为真;如果节点20的状态为假,那么节点51的状态为真;如果节点3的状态为假的话,那么节点52的状态为真。节点1和节点2有一条限制条件注释,表明节点1和节点2不可能同时为真。

表G-1是图G-1转换成决策表的结果。这是通过以下步骤完成的:(1)在图中找出使得每个结果为真的原因组合;(2)将每一种组合在决策表中用一列来表示;(3)对每一个这样的组合给出所有其他结果的状态。在完成这个之后,表G-1的每一列代表一个测试用例。

565

对每一个测试用例来说,表G-1的下部表明了会得到那种结果(用1表示得到对应的结果)。对每个结果来说,造成这一结果的所有原因的组合作表中的列表示。表中的空白表示原因的状态与结果无关。

表G-1 决策表

	测试用例			
	1	2	3	4
原因				
1	1	0	0	
2	0	1	0	
3	1	1		1
结果				
50	1	1	0	0
51	0	0	1	0
52	0	0	0	1

决策表中的每一列都被转换为下面显示的测试用例:

测试用例编号	输入	预期结果
1	D5	显示索引项5
2	L4	显示索引项4
3	B2	无效的命令
4	DA	无效的索引编号

566

因果图可以产生一组有用的测试用例，并能够指出在需求规约中的不完整和不明确之处。在规约详细给出并且输入条件的组合已确定的情况下，它可被用于在任何类型的计算应用程序中产生测试用例。尽管这一技术的人工应用程序是冗长并且有一点复杂的，但是我们可以使用自动化测试工具来自动将需求转换成因果图、决策表和测试用例，详情参见第六部分。

G.8 条件覆盖

条件覆盖是一种白盒测试技术，其测试用例要保证判断中的每个条件的可能结果都至少取过一次。使用这种条件覆盖技术时，无需考虑判断的分支。条件覆盖保证了判断中的每个条件都被覆盖了。然而，这并不需要遍历每个判断的所有真假结果。

使用这个技术创建测试用例的一种方法是建立一张真值表，将所有的条件和它们的反填入表中。如果存在冗余的测试用例则应当删除。

为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个成员的取值范围。

程序：FIELD-COUNT

```

Dowhile not EOF
  read record
  if FIELD_COUNTER > 7 then
    increment COUNTER_7 by 1
  else
    if FIELD_COUNTER > 3 then
      increment COUNTER_3 by 1
    else
      increment COUNTER_1 by 1
    endif
  endif
endif
End_While
End

```

满足条件覆盖的初始测试用例如下所示：

测试用例	值(FIELD_COUNTER)
1	>7, 比如8
2	≤7, 比如7
3	>3, 比如6
4	≤3, 比如3

567

我们可以看到，测试用例2和3是冗余的，可以删除其中的一个，最后得到3个测试用例。

G.9 CRUD 测试

CRUD矩阵，或者说过程/数据矩阵，可以在应用程序开发的分析阶段建立，将数据和过程模型相关联。这保证了数据和过程已经生成并且是经过评估的。它标识并解决了矩阵中的冗余和冲突，细化了数据和过程模型。它将过程与实体相联系，展示出哪个过程创建、读取、更新或者删除了哪个实体的一个实例。

表G-2所示的CRUD矩阵是在开发的分析阶段建立的，在实际系统或者图形用户界面（物理屏幕、菜单等）的设计和开发之前。随着图形用户界面的发展，CRUD测试矩阵被建立，如表G-2所示。这是一个对所有业务对象的生命周期提供检验的测试技术。表G-2中的每个表格项都将被测试。当一个对象没有完整的生命周期操作时，在对应表格项中填入“—”。

表G-2 CRUD测试

对 象	C (通过/失败)	R (通过/失败)	U (通过/失败)	D (通过/失败)	确认删除 (是/否)	测试人员	日期 (月/日/年)
客户	×	×	×	×			
订单	—	×	×	—			
付款	—	×	×	—			
厂商	×	×	×	×			
支票	×	×	—	×			
销售记录	×	×	×	×			
产品	—	×	×	—			
库存	×	×	×	×			
退订	—	×	×	—			
存货	×	—	×	—			
报表	×	×	—	—			
.....							
.....							

另外一个用途就是为阶段性系统测试中的每个操作提供一个单元性能标准。

G.10 数据库测试

下面来描述一下如何进行数据库测试，同时简要介绍一些关系数据库的概念，供测试人员参考。

G.10.1 完整性测试

数据库完整性测试对结构和格式进行检验，遵从完整性的约束条件、业务规则和相互关系，以及关于数据库更新或者对于每个性能约束对数据库进行规范化或反规范化的编辑控制。为了验证数据库的完整性，我们至少要执行6种完整性测试。

1. 实体完整性

568

实体完整性要求每一行必须有一个主键值。举例来说，在一个团队列表中团队ID是主键，那么任何团队都不可以没有团队ID。这个完整性可以使用数据库完整性报告或查询来测试和验证。

2. 主键完整性

每一个主键的值必须是唯一且有效的。举例来说，两个团队不可以共用一个团队ID，而且当要求团队ID是数字值时，像“ABC”这样的团队ID就是错误的。另外一个规则就是主键不可以是空值。这个完整性可以使用数据库完整性报告或查询来测试和验证。

3. 列项完整性

569

每一列中的值也要遵循该列特定的规则。举例来说，表示团队成员数量的这一列中的值必须是正整数并且不能超过7。这个完整性可以使用数据库完整性报告或查询来测试和验证。也可以通过如下测试技术进行验证：范围测试、边界值测试、字段完整性测试、正负测试。

4. 域完整性

域是由一系列描述一些值的数据和它们的特性所组成的一个对象。举例来说，“日期”是一个定义好的基本数据类型，有自己的字段长度、格式和验证规则。我们可以基于域来定义列，比如，我们可以定义一列是订单日期。这个完整性可以使用数据库完整性报告或查询来测试和验证。也可以通过如下技术进行验证：范围测试、边界值测试、字段完整性测试、正负测试。

5. 用户定义的完整性

用户定义的完整性检查是在标准的行列检查之外的特定的检验规则。用户为特定的数据项定义的规则往往需要使用过程语言进行手工编写。

有一个替代写程序的方法就是使用断言，如果有的话。断言是一种不与特定的行或列相关联的独立检查，它是自动应用的。

6. 参照完整性

主键是用来唯一标识一个特定实体的候选键。在一个团队表中，团队编号可以作为主键。外键是在另外一个实体中引用主键的键，作为一个引用。举例来说，团队ID对于一个队员实体来说是一个外键，但是对于团队实体来说就是主键。

一个表有自己的业务规则来管理它里面实体之间的关系。举例来说，队员必须且只能属于一个团队，而一个团队在某一时刻可能没有队员、有一个队员或者有多个队员。这要根据实体之间的关系来决定。任何在系统中“游荡”（没有所属团队）的成员都是无效的，对于这样的记录我们认为它是一个“孤儿”。

举例来说，假设一个团队可以没有、有一个或者有多个队员，但是任何队员都不可以没有所属团队。表G-3中所示的测试用例可以用来验证参照完整性。

表G-3 参照完整性测试用例

测试用例	期望结果
1. 插入一个团队	无任何队员实体与之关联
2. 插入一个队员	有一个与团队实体相关联的外键
3. 尝试删除一个有与队员实体相关联的外键的团队实体	无法自动执行删除操作，给出确认提示
4. 更新一个队员	与团队实体相关联的团队外键仍然存在

下面是另外一些数据库测试方法。

- 控制测试——包括一系列需要测试的控制问题。
 - 安全性测试——保护数据库不受非法入侵的破坏。
 - 备份测试——验证备份系统的能力。
 - 恢复测试——在发生错误导致系统不可靠时，能够将数据库恢复到一个先前的正确状态。
 - 并发测试——保证诸如查询和更新这样的并发过程不会互相干扰。
 - 死锁测试——保证两个并发过程不会形成死锁，能够通过适当的完成机制来互相排斥。
- 数据内容验证——周期性地进行的审计并与已知参照源进行比较。
- 刷新验证——对刷新数据库和数据转换的外部系统进行验证。
- 数据使用——包括验证数据库的编辑和更新。很多时候，开发人员会为实体的列创建过多或者包含过少的字符。测试人员往往需要将图形用户界面字段的实际输入字符数与对应的实体字段长度进行比较，以验证它们是一致的。提示：一种确保数据库列长度满足要求的方法是使用Windows的复制编辑功能将一个非常大的文档复制粘贴到图形用户界面的每个字段中。一些可以用来生成数据的测试技术如下：范围测试、边界值测试、字段完整性测试、正负测试。大部分数据库都提供查询功能，以便测试人员验证对数据的更新或编辑是否正确。
- 存储过程——由应用程序的特定事件来存储和调用这些过程。

G.10.2 数据建模基础

这部门内容帮助读者熟悉数据建模的概念和术语，以便能够针对关系设计完成数据库和图形用户界面字段测试（参见G10节、G26节、G22节和G36节）。同时，这也是对测试中的关系数据库的一个有用的讲解。

1. 什么是模型

模型是对真实世界中系统的一个简化描述，可以用来进行估算和预测。只有用户感兴趣的方面才会加到模型中，其他的都被省略掉了。

建立模型要用到3种材料，即金属、木材和泥土。

我们将选择最合适的材料来建立模型，虽然这种材料可能和建立实际系统使用的材料不同。系统的详细规约往往被用作真实世界的一个模型。

我们认为一个模型包括以下两个特征：结构和内容。

模型的结构反映了系统不变的方面，而内容则反映了系统动态的方面。举例来说，一个气象预报模型的结构是公式化的，而内容是由一段时间内收集起来的数据（温度、湿度、风速和大气压力）组成的。

2. 为什么要建立模型

我们必须要有办法来测量现实世界中的系统，这样才能理解并有效地使用它们，监控它们的性能，以及预测其未来的性能。一般情况下，我们不可能去测量实际的系统，这可能过于昂贵或者过于危险。飞机制造商在将一位飞行员送上新飞机之前，必须保证飞机能够飞行。而汽车制造商也希望能够在启动装配流水线之前先知道汽车看起来会怎么样。

572

我们有理解、测量、控制一个现实世界中的系统（用户业务）这样的需求。想要对业务系统进行及时、低成本的测量和预测，最简单的方法就是建立业务模型。对我们的模型来讲，数据是最恰当的材料，因此有了“数据模型”这个名称。数据模型的结构应该能够表现出用户业务中很少随着时间改变的方面，模型的内容（存储在模型内部的值）则代表那些会随时间变化的信息。这样一来，我们就得到了一个结构稳定且易于维护的数据模型。

我们建立的应用程序负责对模型的内容进行添加、修改和删除，同时也能够给出关于内容的报告。

下面是使用这项技术的好处。

- 数据模型相对稳定的方面允许我们更好地改变业务需求。业务的变更经常会导致内容维护和报告方式的改变。对比起来模型的结构发生改变的情况就很少了。
- 使用这项技术，我们创建的数据模型将会独立于当前业务过程（不是业务规则）和当前的数据处理技术。
- 这项技术的另外一个好处就是可以用于一些当前面向过程的技术无法起作用的情况。举例来说，在信息管理应用程序中并没有明确定义的过程。用户无法确切地指定如何使用数据。通过创建一个其结构反映了业务结构的数据模型，我们可以支持任何合理的对数据的使用。
- 数据分析开始于数据模型的开发。

G.10.3 表的定义

表是一个将事实、数字以及类似的东西系统地记录在很多列中的一个清单。

我们需要对信息进行整理以便存储或者展示时，就需要使用表。表能够相对容易地使用一种清晰、明确和简单的格式来对信息进行创建、维护和展示。我们将会遇到下面几种表：

- 内容表；
- 度量转换表；
- 重量和测量表；
- 税款表。

表G-4举例说明了表的特征。

表G-4 表样本

姓 名	地 址	电话号码
Bill Smith	3290 Oak Lane, Dallas, Texas	(972)329-6723
Joe Jones	129 Cliff Avenue, Austin, Texas	(812)456-2198
Sue Maddox	1421 Millington Drive, Boca Raton, Florida	(305)402-5954
Jerry Jones	112 Cowboys Drive, Portland, Oregon	(265)693-2319

1. 表名称

573

一个表用它的名称来标识，因此它的名称在业务的范围内必须是唯一的。

2. 列

一个表被竖直地分成很多列。在一个指定列中的所有条目都是同一类型的并且有相同的格式。一列包含了表中所有行中的数据的一个单独的部分。每一列必须有一个在表中唯一的名称。表名称和列名称的组合在业务里面是唯一的,如客户.姓名、客户.编号、员工.姓名、员工.编号等。

3. 行

一个表被水平地分成很多行。每一行必须有一个唯一的标识。每一行由相同数目的单元项组成,包含不同类型和格式的一块数据。

4. 次序

表中行和列的次序是任意的。也就是说,行和列排列的次序对于数据的意义是没有影响的。事实上,表的每个用户对于行和列的次序都有着独特的需求。因此,次序必须是没有特殊意义的。

基于以上这些定义,对于记录数据需求而言表是非常有用的。它们能够很容易地被开发人员和使用者所理解。

我们定义一个表来表示我们的模型中的每个对象。表的列提供了关于对象的描述信息,而行则给出了对象的每个实例。

G.10.4 实体的定义

实体是一个唯一标识的人、地点、事物或者用户所关心的事件,也是应用程序维护和报告数据的对象。

574

当我们建立一个数据模型时,必须首先确定要包括哪些真实世界中的对象。我们只包括那些用户感兴趣的对象。而且,我们只包括那些计算机应用程序所需要的对象。

我们将这些对象(实体)分组,叫做实体类型。举例来说,时装商店建立数据模型时可能分出如下实体类型:客户、销售的产品、产品供应商。然而这种分组对于真实世界的有用模型是不恰当的。依据商店所卖时装的种类,客户可能希望能够按照风格、类型、尺码、颜色等方面来对产品进行分类。由于真实世界中的很多模糊定义,对象所属种类的辨别是很困难的。在我们的模型中,我们必须有明确的定义,因此,只将其中每个实例都可以进行唯一标识的对象组定义为实体类型。

每个实体类型都有一个唯一的名称,如客户、供应商、员工等。

G.10.5 识别主键

每个实体都必须有一个主键。

为了能够唯一标识实体类型的每个实例,我们必须定义一个叫做主键的键值。它的值由用户或者应用程序指定。主键可能有多种选择方法。对于员工这个实体类型来讲我们可能选择社会保险编号或者创建一个员工编号。主要的需求是每个值必须是唯一的。同时用户要能够通过主键来标识出实体的一个实例,这也是非常重要的。我们选择的主键也应该是比较稳定的,不应经常发生变动,而且要尽可能短。这也是序列号一直很流行的原因,因此它们是一次性分配的,不会改变,而且是唯一的。(注意:在现实世界中,可能会不小心分配出重复的序列号。)

注意：键并不是访问路径，它只是一个唯一标识。

1. 复合主键

主键可能由多列组成。例如，汽车能够通过如下组合唯一标识：品牌+型号+车身号。由多列组成的键就是一个复合键。

2. 空值

575

在实体的任何描述信息中，有可能出现某个特定实例的某些数据不存在这种情况。举例来说，当一个员工描述第一次添加到应用程序中时，该员工的部门编号和电话号码可能是不知道的。这时我们就希望看到一个未知的值，也就是空值。在应用程序中，我们可能使用空格、零或者一些特殊标识符来代表空值。因为空值代表了未知，所以空值不能用于比较（例如，进行等于比较），同样也不能用来进行数值计算，因为这样做的结果也是未知的。在我们的数据模型中，我们要指出哪些列可能包含空值。

我们在此提出这个概念是因为如下规则：主键不可以是空值。

这是非常重要的。空值代表了我们不知道确切的值是多少，但是主键必须是确定的值，这样才能唯一地标识与它相关的实体类型的每个实例。在复合键中，可能有些列中会有空值，但是不可以所有列都为空值。

3. 定义实体

考虑如下清单：

- 哪个是实体类型？
- 哪个是实体的实例？
- 哪个以上都不是？
- 哪个可以成为合适的键？
 - 汽车。
 - 福特。
 - 超人。
 - Nietzsche。
 - 电话。
 - 电话号码。
 - 房屋。
 - 邮政编码。
 - 碧绿色。
 - 七。
 - 婚姻。

在试图将以上列表中各项分类为实体类型或者实例的过程中，我们发现应用的环境是很重要的。以汽车为例。如果用户是汽车经销商，那么汽车就是实体类型。然而，如果用户正在试图记录交通工具的种类，那么汽车就是实体的实例。Ford（福特）可能是汽车的制造商、美国总统，也可能是过河的一种方法。

在实际情况下，满足参照完整性规则意味着以下几点。

- 可以插入一名客户而不需要进行完整性检查。
- 可以插入一种产品而不需要进行完整性检查。
- 可以插入一个订单，但是客户编号外键必须存在于客户实体中而且产品代码外键必须存在于产品实体中。
- 如果其主键存在于订单实体中并且是它的一个外键，不可以删除一名客户。
- 如果其主键存在于订单实体中并且是它的一个外键，不可以删除一种产品。
- 可以更新一个订单，但如果客户和产品属性的值被修改了，客户编号外键必须存在于客户实体中而且产品代码外键必须存在于产品实体中。

有时，要满足完整性规则可能会更加复杂。例如，在输入订单的时候我们可能需要允许创建一个客户，在这种情况下应用程序代码的编写必须增强一个已经修改过的规则：可以插入一个订单，但是客户编号外键必须存在于客户实体中，或者客户编号外键必须在订单插入的时候同其属性一起插入。产品代码外键必须存在于产品实体中。

如果这些约束看起来过于严格了，试问你是否希望一个销售人员为并不存在的客户和产品输入订单。

完整性约束也可应用于实体子类型上。在某种程度上，一个子类型仅仅简单地与另外一个实体之间有一个专门的关系（1:1），其中，子类型的主键也是另一个实体类型的外键。换句话说，除非Joe Bloggs已经是一位员工，否则我们是不能将他任命为市场代表的。参照完整性规则必须作为数据模型的一部分记录在文档中。

在这种情况下，基于依赖性约束的规则应该如下所述。

- 可以插入一个订单而不需要进行依赖性检查（尽管不插入至少一个订单行这样做是没有意义的）。
- 可以插入一个订单行项，但是订单编号外键必须存在于订单实体中或者必须与其属性一起插入订单行。
- 可以删除一个订单行而不需要进行依赖性检查。
- 除非一个订单的所有依赖订单行都已经预先删除掉了，否则是不能删除这个订单的。
- 删除订单必须触发所有相关依赖性订单行的删除。

依赖性约束必须作为数据模型的一部分记录在文档中。

596

G.10.11 依赖性约束规则

除非依赖性实体所依赖的实体已经存在了，否则它是不能存在的。

依赖性约束规则是可以应用于依赖实体的参照完整性约束的一种特殊形式。一些数据库管理系统自动增强了大多数的依赖性约束，而其他的系统没有这样。

表G-30和表G-31说明了一个具有多个行项的订单的依赖性。

表G-5 员工表

员 工	
员工编号	工位编号
PK	FK
	ND
12345	004
23456	003
98751	001

表G-6 工位表

工 位	
工位编号	员工编号
PK	FK
	ND, NL
003	23456
001	98751
002	-NULL-
004	12345

员工编号这一列中的PK代表了此列是主键，工位编号这一列中的FK代表了此列是外键（也就是另外一个表的主键）。此列的ND强调了这是一对一的关系，没有任何重复的值（同一个工位不可以分配给两个不同的员工）。

表G-6显示了同样的关系。ND代表了一个员工不可以分配到两个工位。注意，这个表中的员工编号列中有NL这个标识，它代表了可能有的工位是未分配的。

尽管两种方法都可以表示这一关系，但是有一些指导方针还是要遵守的。

- 不要同时使用两种方法来记录这一关系，选择其一。
- 选择能够减少或者消除记录空值这种方法的情况。注意，这表明要选择出现次数最少的情况作为实体的外键。

基于以上的方针，我们这个例子中的关系的最好表示方法如表G-5所示，将工位编号作为员工的外键（尽管表G-6也是正确的）。

3. 一对多

一对多关系是最常见的关系，其记录技术也很简单。一对多关系将一个实体类型的一个实例和另一个实体类型的零个或多个实例相关联。

举例来说，我们再来看上面描述的那家公司。当为员工分配电话时，公司就没有那么开明了。每个员工必须和其他员工共用一个电话号码和线路。表G-7和表G-8展示了电话号码和员工之间的关系。

表G-7 员工表

员 工	
员工编号	电话号码
PK	FK
12345	1111
23456	1954
98751	2654

表G-8 电话线路表

电话线路	
电话号码	
PK	
1954	
2222	
1111	
2654	

这种关系的记录看起来和一对一关系的记录很相似。然而，表示一对多关系只有一种方法。

在这里我们发现许多属性都是不依赖于键的，单位价格依赖于合同类型和产品编号，总价依赖于单位价格和数量。

精简到第三范式需要创建一个新实体产品/型号/合同，它的键是产品编号加型号加合同类型，单位价格是这个实体的一个属性。总价是由其他属性的量计算得来的，并且可以从表中删除，需要时也可以计算出来，这被称为一个派生属性。

第三范式应该看起来像表G-25和表G-26所示的那样。

表G-25 订单表

订 单					
订单编号	产品编号	型 号	客户编号	合同类型	数 量
PK	FK	FK		FK	
XN223	4068	067	112339	EMPLOYEE	1
XQ440	4068	067	990613	INTERNAL	5
XL715	4068	067	574026	DEALER	20
XB229	5160	020	390651	RETAIL	2
ZC875	5360	B40	740332	BUSINESS	1
YS8/13	5360	B40	468916	GOVERN'T	4

表G-26 产品/型号/合同表

产品/型号/合同			
产品编号	型 号	合同类型	单位价格
PK			
4068	067	EMPLOYEE	1 098美元
4068	067	INTERNAL	875美元
4068	067	DEALER	1 170美元
5160	020	RETAIL	2 960美元
5360	B40	BUSINESS	33 600美元
5360	B40	GOVERN'T	28 400美元

在这种范式中，价格和数量是可以改变的。两个实体当中的数据联合到一起才能计算出总价。为了保护客户不受价格改变的影响，需要对模型做哪些改变呢？如果决定将总价作为订单实体的一个属性，对应用程序又将产生何种影响呢？

4. 模型提炼

这一部分讨论了可以被合并入一个数据模型的附加提炼过程。

关于这些提炼过程，重要的是它们给模型引入了限制，在设计过程中应当将这些限制记录到文档中并合并入应用程序。

5. 实体子类型

我们经常必须要将一个已定义的实体分解成子类型。

表G-12 员工表

员 工
员工编号
11111
22222

表G-13 项目表

项 目
项目编号
ABCD
WXYZ

表G-14 员工/项目表

员工/项目	
员工编号	项目编号
11111	ABCD
11111	WXYZ
22222	ABCD
22222	WXYZ

我们可以看到员工11111和两位员工/项目实例相关联（11111ABCD和11111WXYZ）。这两个员工/项目实体中的每一个都与一个项目实体相关联，结果就是每位员工的实例通过员工/项目实体都可能跟多个项目实例相关联。通过同样的技术，每个项目的实例也能够跟多个员工实例相关联。

5. 多重关系

有时在同一实体类型的事件之间会有不只一种关系。遇到这种情况时，应当将每一个关系单独地标识出来，并将它们记录在文档中。例如，在上一个例子中，有可能需要将分配到每个项目的项目领导独立于其他员工进行记录。这一关系可能是一对多关系，其中的项目主管员工编号是项目表的外键。

6. 实体与关系

实体与关系之间并不总是容易区分的。考虑下面的例子。

一个客户从经销商那里买了辆汽车，这一交易是与经销商的一位销售人员磋商的结果。这个客户以前可能从这个经销商这儿买过汽车，但可能是与另外一位销售人员打的交道。

购买是一个客户与销售人员之间的关系吗？它是一个与客户、销售人员、汽车相关的实体吗？如何看待这样一个真实世界中的情况经常是随意的，没有正式的规则可以引导你。幸运的是，我们用于将实体和关系记录在文档中的技术可以减少和消除这一问题。

如果我们认为购买协议是一个实体的话，我们要选择一个主键，比如说协议编号，并且定义它与其他实体的关系。在销售人员和购买协议之间是一对多关系，而且如果我们让客户很满意的话，客户和购买协议之间也可能是一对多关系。通过将客户编号和员工编号置为购买协议的外键，我们将这些关系记录在表G-15当中。

如果不将购买协议看做一个实体，就必须将客户和销售人员的之间的关系记录在文档中（见表G-16）。因为在一般的情形下，客户与销售人员的之间的关系是多对多的，我们必须创造一个新的实体，即客户/销售人员。这个实体具有一个由客户编号和员工编号组成的复合键。我们可能必

我们创建了一个新实体（客户地址），它有一个由客户编号和行号组成的混合键（为了标识出客户地址的每一行）。这一新实体是依赖于客户实体的，而且它允许一个地址具有可变的行数（0~99）。

一般来说，多值属性是可以被标识出来的，因为它们被记录为数组，包括结构数组，在结构数组中每个元素事实上都是属性的一个不同值。在某些情况下，比如上面的例子，通过使用不同的列名掩饰了一个属性是多值的这一事实。名字的相似性让我们看到了这一事实。下面是一些例子。

- 当前销售人员，上一销售人员。
- 第一部门办公室，第二部门办公室。

2. 第二范式（2NF）

每个属性都依赖于整个键。

规范化的第二个步骤是移除部分依赖于主键的属性并将它们变成新的实体。

让我们看一下表G-21，表中实体（产品型号）是一个由所有产品及其型号组成的实体，其键是产品编号加型号。让我们假设每个产品都有一个单独的货源供应商并且知道每种型号现有的量是非常必要的。

表G-21 产品型号表

产品型号					
产品编号	型 号	产品描述	型号描述	现有的量	货源供应商
PK					
3084	032	4_PLEX CPU	SMALL MEMORY	3	FUJISAWA
3084	064	4_PLEX CPU	MORE MEMORY	2	FUJISAWA
3084	0C8	4_PLEX CPU	OODLES OF MEMORY	0	FUJISAWA
3180	001	TERMINAL	TWINAX CONNECTION	55	DALLAS
3180	002	TERMINAL	COAX CONNECTION	83	DALLAS
3274	A41	CONTROL UNIT	BIG MODEL(LOCAL)	15	SAO PAULO
3274	C41	CONTROL UNIT	BIG MODEL(REMOTE)	29	SAO PAULO
SAO PAULO	C61	CONTROL UNIT	DESK TOP MODEL	11	

表G-22中的产品描述和源供应商在生成表G-23中的产品型号实体时，去除了表G-22中的产品描述和货源供应商，只保留了产品编号。

表G-22 产品表

产 品		
产品编号	产品描述	货源供应商
PK		SUPPLY
3084	4_PLEX CPU	FUJISAWA
3180	TERMINAL	DALLAS
3274	CONTROL UNIT	SAO PAULO

在我们的例子当中，最为安全的方法是将客户编号的和部门编号域分开定义。

提示：在定义字段（如客户编号、员工编号和部门编号）的域时要谨慎。我们很自然地认为这些字段是数值的，所有的分配值可能都是数值的。然而，字符也是经常会出现标识中的。

G.10.9 域名

每个域应该有一个在组织中唯一的名字。这个名字和在域中定义值的规则应当被记录在文档中。基于一个域的单独一列主键通常有与域名一样的名字（如客户编号）。如果一个键（主键或外键）是混合的，每一列通常都具有与域一样的名字。当一个域被一个实体的属性引用多次时（例如，出生日期、一个员工的入职日期），该域的域名应当是属性列名的一部分。

域以及基于它们的属性必须是不可分解的。

上面的叙述并不意味着属性不会随时间而改变。考虑下面这个可分解的属性和域的例子。

只要记录了一个订单，就应当给这个订单分配一个订单编号。订单编号根据以下规则创建。

(1) 客户编号组成了订单编号的第一个（最高阶的）部分。

(2) 订单创建日期以YYMMDD（年月日）的格式作为订单编号的第二个部分。

(3) 订单编号的最后两位是一个保证唯一性的序列编号，以免一个客户在一天当中提交多个

585 订单时，会造成编号的唯一性被破坏。

因为使用了订单编号这个项，我们会希望将其作为一个单独的列。但不要这么做。这个例子中的主键是由客户编号、订单创建日期和一个序列编号组成的混合键。组成这个混合键的每一个属性都基于一个不同的域。现在我们可以看到订单中可能有一个属性是基于客户编号的域的。该域的规则的任何变化都会对订单实体产生影响。

已经说过所有的域都是不可分解的，现在我们给出两个例外。

(1) 日期。

(2) 时间。

日期通常是以月/日/年的格式出现的，通常没有必要将日期记录为3个单独的属性。类似地，时间可以以小时/分钟的形式记录。

1. 属性与关系

正如在实体和关系之间进行选择有一定的任意性一样，在属性与关系之间做出选择也是类似的。可以把属性看做是该属性有效值的表的外键。例如，如果需要将眼睛的颜色作为一个员工的属性记录下来，可以建立一个实体，叫做颜色，其主键是颜色的名称，然后可以在员工中建立眼睛颜色作为一个外键。这也许并不能提供比一个简单的属性更多的优势，甚至还会遇到一些问题，如果选择将头发颜色作为一个与同一主键相关的外键加入头发颜色表中，可能会遇到一个蓝头发、红眼睛的员工。

尽管下面的例子看起来可能是很平常的，真实世界中的选择通常更加微妙。如果希望有一个用于编辑检查的表或者需要一个关于报告的很长的描述，通常你可能选择一个外键而不是一个简单的属性。这个描述可能是表中的一个属性，在这个表中该外键是一个主键。

2. 什么是规范化

规范化是一个将一组初始的实体精炼成为一个最优模型的过程。目的是消除数据冗余并确保

数据结构尽可能具有灵活性、易理解性和可维护性。

规范化是通过确保一个实体仅仅包括那些依赖于该实体的键的属性而达成的。这里的依赖意思是键的每个值都仅仅决定了该实体的每个属性的一个值。如果这个概念不清楚，不要灰心，本节的后面我们会解释的。换一个说法，规范化意味着确保：每个属性都依赖于该键，每个属性都依赖于整个键，每个属性都仅仅依赖于这个键。

3. 非规范化实体的问题

表G-17论述了试图维护一个非规范化实体的过程中可能出现的问题。表G-17中的例子是非规范化的，因为部门名称依赖于部门编号，而不是员工编号，而员工编号才是实体的键。考虑一下这一设计在应用程序中产生的影响。

- 修改异常：假设一个业务重组必须将部门354的名称变成“广告和促销部”，这时我们需要一个专门的程序来准确地、完整地修改这一信息，并且在数据库中每次出现该数据时都加以修改。
- 插入异常：部门220雇用了一位新员工。维护数据的员工可能还没有获得所有的相关信息，那么，他要么得扫描数据查找部门220中现有的名字，要么很可能要猜想并且把我们的新员工分配到部门名叫做“SHALLOWTHOUGHT”的部门220。这个部门现在的名字到底叫什么呢？
- 删除异常：编号00215的员工退休了，代替她的人下周开始工作。但是，如果删掉员工00215对应的项，我们就会失去能够告诉我们出版部门存在的信息。
- 冗余：通过设计考虑到这些异常存在的程序，可以减小这些异常的影响。一般来说，这会导致代码比原本需要的更加复杂，并且需要附加代码来解决这些不一致的地方。这些增加了开发和维护的成本却没有消除问题。更有甚者，数据的重复会增加文件和数据库的大小，并且导致操作该应用程序的成本增加。

我们将以上所有问题合称为更新异常。

表G-17 员工表

员 工				
员工编号	名 字	薪 水	部 门	部门名称
PK				
00100	CODD, E.F.	65 736	220	DEEP THOUGHT
00135	KENT, W.	58 200	220	DEEP THOUGHT
00171	LEWIS, W.	49 900	220	DEEP THOUGHT
00190	SMITH, S.	64 000	220	DEEP THOUGHT
00215	DATE, C.J.	51 500	114	PUBLISHING
00529	FLAVIN, M.	35 700	354	ADVERTISING
00558	CLARK, G.	33 600	354	ADVERTISING

G.10.10 规范化的步骤

我们通过以下一系列例子的讨论来解释规范化的过程，这些例子阐明了将非规范化数据精减成第三范式所需要的三个基本步骤。

1. 第一范式 (1NF)

每个属性都依赖于该键。

每个属性只能从对应的键的取值范围中取一个值。规范化的第一个步骤就是删除从对应键中取了多个值的属性，将这些属性变成新的实体。

例如，考虑如下实体（客户），其键属性是客户编号，如表G-18所示。

表G-18 客户表

客 户						
客户编号	分店	CR CD	客户类型	地址第1行	地址第2行	地址第3行
PK						
003531	0059	A	C	JOHN BLOGGS	25 MAIN ST.	DALLAS,TEXAS
094425	0047	B	C	SAM HOSER	19 REDUNDANT	HOUSTON,TEXAS
976531	0099	I	I	IBM DEPT 344	3500 STORY	BOCA RATON,FLORIDA

在这个案例中，多值属性由3个“属性”组成，地址第1行、地址第2行、地址第3行。事实上，这些实际上是一个数组中的3个元素，这一实体的第一范式如表G-19和表G-20所示。

表G-19 客户表

客 户			
客户编号	分店	CR CD	客户类型
PK			
003531	0059	A	C
094425	0047	B	C
976531	0099	I	I

表G-20 客户地址表

客户地址		
客户编号	行号	地址行
PK		
003531	01	JOHN BLOGGS
003531	02	25 MAIN ST.
003531	03	DALLAS,TEXAS
094425	01	SAM HOSER
094425	02	19 REDUNDANT
094425	03	HOUSTON,TEXAS
976531	01	SALES DEPT 355
976531	02	3500 STORY
976531	03	BOCA RATON,FLORIDA

我们创建了一个新实体（客户地址），它有一个由客户编号和行号组成的混合键（为了标识出客户地址的每一行）。这一新实体是依赖于客户实体的，而且它允许一个地址具有可变的行数（0~99）。

一般来说，多值属性是可以被标识出来的，因为它们被记录为数组，包括结构数组，在结构数组中每个元素事实上都是属性的一个不同值。在某些情况下，比如上面的例子，通过使用不同的列名掩饰了一个属性是多值的这一事实。名字的相似性让我们看到了这一事实。下面是一些例子。

- 当前销售人员，上一销售人员。
- 第一部门办公室，第二部门办公室。

2. 第二范式（2NF）

每个属性都依赖于整个键。

规范化的第二个步骤是移除部分依赖于主键的属性并将它们变成新的实体。

让我们看一下表G-21，表中实体（产品型号）是一个由所有产品及其型号组成的实体，其键是产品编号加型号。让我们假设每个产品都有一个单独的货源供应商并且知道每种型号现有的量是非常必要的。

表G-21 产品型号表

产品型号					
产品编号	型 号	产品描述	型号描述	现有的量	货源供应商
PK					
3084	032	4_PLEX CPU	SMALL MEMORY	3	FUJISAWA
3084	064	4_PLEX CPU	MORE MEMORY	2	FUJISAWA
3084	0C8	4_PLEX CPU	OODLES OF MEMORY	0	FUJISAWA
3180	001	TERMINAL	TWINAX CONNECTION	55	DALLAS
3180	002	TERMINAL	COAX CONNECTION	83	DALLAS
3274	A41	CONTROL UNIT	BIG MODEL(LOCAL)	15	SAO PAULO
3274	C41	CONTROL UNIT	BIG MODEL(REMOTE)	29	SAO PAULO
SAO PAULO	C61	CONTROL UNIT	DESK TOP MODEL	11	

表G-22中的产品描述和源供应商在生成表G-23中的产品型号实体时，去除了表G-22中的产品描述和货源供应商，只保留了产品编号。

表G-22 产品表

产 品		
产品编号	产品描述	货源供应商
PK		SUPPLY
3084	4_PLEX CPU	FUJISAWA
3180	TERMINAL	DALLAS
3274	CONTROL UNIT	SAO PAULO

表G-23 产品型号表

产品型号			
产品编号	型 号	型号描述	现有的量
PK			
FK			
3084	032	SMALL MEMORY	3
3084	064	MORE MEMORY	2
3084	0c8	OODLES OF MEMORY	0
3180	001	TWINAX CONNECTION	55
3180	002	COAX CONNECTION	83
3274	A41	BIG MODEL(LOCAL)	15
3274	C41	BIG MODEL(REMOTE)	29
3274	C61	DESK TOP MODEL	11

旧的产品型号实体现在依赖于新的产品型号实体,可以加入新的型号而不需要保留产品描述和源供应商信息,可以删除型号同时仍保留关于产品本身的信息。

当混合键标识一个实体类型的实例时,属性部分依赖于键的情况就变得明显了。

如果一个产品有多个源供应商时,会对以上实体造成什么样的影响呢?

3. 第三范式 (3NF)

每个属性都仅仅依赖于这个键。

规范化的第三个步骤就是去除依赖于实体的其他非主属性的属性。

在这一点上应当指出,非主属性是一个既不是主键也不是候选键的属性。候选键是也能够唯一标识出一个实体的每个实例的属性,但它不是主键。(例如,一个人事文件的主键是员工的编号,这一文件也包括一个社会保险编号,这两个键都可以唯一标识该员工,这时员工的编号就是主键,社会保险编号就是候选键。)

考虑表G-24中的订单实体,其每一个实例都代表了一种产品的一个订单,作为已知量,假设单位价格随着机器以及合同的变化而变化。

表G-24 订单表

订 单							
订单编号	产品编号	型 号	客户编号	合同类型	单位价格	数 量	总 价
PK	FK	FK					
XN223	4068	067	112339	EMPLOYEE	1 098美元	1	1 098美元
XQ440	4068	067	990613	INTERNAL	875美元	5	4 375美元
4068	4068	067	574026	DEALER	1 170美元	20	23 400美元
XB229	5160	020	390651	RETAIL	2 960美元	2	5 920美元
ZC875	5360	020	740332	BUSINESS	33 600美元	1	33 600美元
YS8/13	5360	B40	468916	GOVERN'T	28 400美元	4	113 600美元

在这里我们发现许多属性都是不依赖于键的，单位价格依赖于合同类型和产品编号，总价依赖于单位价格和数量。

精简到第三范式需要创建一个新实体产品/型号/合同，它的键是产品编号加型号加合同类型，单位价格是这个实体的一个属性。总价是由其他属性的量计算得来的，并且可以从表中删除，需要时也可以计算出来，这被称为一个派生属性。

第三范式应该看起来像表G-25和表G-26所示的那样。

表G-25 订单表

订 单					
订单编号	产品编号	型 号	客户编号	合同类型	数 量
PK	FK	FK		FK	
XN223	4068	067	112339	EMPLOYEE	1
XQ440	4068	067	990613	INTERNAL	5
XL715	4068	067	574026	DEALER	20
XB229	5160	020	390651	RETAIL	2
ZC875	5360	B40	740332	BUSINESS	1
YS8/13	5360	B40	468916	GOVERN'T	4

表G-26 产品/型号/合同表

产品/型号/合同			
产品编号	型 号	合同类型	单位价格
PK			
4068	067	EMPLOYEE	1 098美元
4068	067	INTERNAL	875美元
4068	067	DEALER	1 170美元
5160	020	RETAIL	2 960美元
5360	B40	BUSINESS	33 600美元
5360	B40	GOVERN'T	28 400美元

在这种范式中，价格和数量是可以改变的。两个实体当中的数据联合到一起才能计算出总价。为了保护客户不受价格改变的影响，需要对模型做哪些改变呢？如果决定将总价作为订单实体的一个属性，对应用程序又将产生何种影响呢？

4. 模型提炼

这一部分讨论了可以被合并入一个数据模型的附加提炼过程。

关于这些提炼过程，重要的是它们给模型引入了限制，在设计过程中应当将这些限制记录到文档中并合并入应用程序。

5. 实体子类型

我们经常必须要将一个已定义的实体分解成子类型。

(1) 定义。实体子类型的定义如下。

- 实体子类型具有对于该子类型来说特有的属性；
- 实体子类型参与了对于该子类型特殊的关系；
- 实体子类型通过实体的键的一个子集标识。

实体子类型和一个依赖的实体是不一样的。依赖的实体是通过一个混合键标识的，这个混合键由主实体的键加上附加的有资格的属性组成。

当然这也不是一定的，有时实体子类型具有与主实体同样的键而且仅仅是该实体的一个子分类。

例如，假设我们全都是员工，也就是员工实体的实例。但是一些员工也是市场代表，具有对他们的职位而言独一无二的属性（市场部门、团队、代理区域、配额等）。市场代表实体是员工实体的一个子类型。

员工实体的附加子类型可能有以下几种。

- 作为经理的员工。
- 作为股东的员工。
- 作为受益者的员工。

实体子类型的存在会产生参照完整性的问题，这一问题在下一小节中讨论。

(2) 参照完整性。完整性规则：对于表中任意外键的值，必须有一个对应的同一个表或另外一个表的主键值。

如上所述，完整性规则是非常简单的。增强这一规则可能需要很多复杂的应用程序代码，在此之前还需要进行重要的设计工作。幸运的是，一些数据库管理系统具有一些内置特性，大大简化了参照完整性的规则（例如，IMS/VS中的逻辑插入、重置和删除规则）。

表G-27、表G-28和表G-29通过3个实体（客户、产品和订单）的方法论证了这一问题。

表G-27 客户表

客 户
客户编号
PK
221356
840723
737174

表G-28 产品表

产 品
产品代码
PK
3084XC8
4260067
5360A23

表G-29 订单表

订 单		
订单编号	客户编号	产品代码
PK	FK	FK
ZA8845	221356	4260067
YB4320	737174	3084XC8
XN7691	840723	4260067
ZL3940	221356	5360A23

594

595

在实际情况下，满足参照完整性规则意味着以下几点。

- 可以插入一名客户而不需要进行完整性检查。
- 可以插入一种产品而不需要进行完整性检查。
- 可以插入一个订单，但是客户编号外键必须存在于客户实体中而且产品代码外键必须存在于产品实体中。
- 如果其主键存在于订单实体中并且是它的一个外键，不可以删除一名客户。
- 如果其主键存在于订单实体中并且是它的一个外键，不可以删除一种产品。
- 可以更新一个订单，但如果客户和产品属性的值被修改了，客户编号外键必须存在于客户实体中而且产品代码外键必须存在于产品实体中。

有时，要满足完整性规则可能会更加复杂。例如，在输入订单的时候我们可能需要允许创建一个客户，在这种情况下应用程序代码的编写必须增强一个已经修改过的规则：可以插入一个订单，但是客户编号外键必须存在于客户实体中，或者客户编号外键必须在订单插入的时候同其属性一起插入。产品代码外键必须存在于产品实体中。

如果这些约束看起来过于严格了，试问你是否希望一个销售人员为并不存在的客户和产品输入订单。

完整性约束也可应用于实体子类型上。在某种程度上，一个子类型仅仅简单地与另外一个实体之间有一个专门的关系（1:1），其中，子类型的主键也是另一个实体类型的外键。换句话说，除非Joe Bloggs已经是一位员工，否则我们是不能将他任命为市场代表的。参照完整性规则必须作为数据模型的一部分记录在文档中。

在这种情况下，基于依赖性约束的规则应该如下所述。

- 可以插入一个订单而不需要进行依赖性检查（尽管不插入至少一个订单行这样做是没有意义的）。
- 可以插入一个订单行项，但是订单编号外键必须存在于订单实体中或者必须与其属性一起插入订单行。
- 可以删除一个订单行而不需要进行依赖性检查。
- 除非一个订单的所有依赖订单行都已经预先删除掉了，否则是不能删除这个订单的。
- 删除订单必须触发所有相关依赖性订单行的删除。

依赖性约束必须作为数据模型的一部分记录在文档中。

596

G.10.11 依赖性约束规则

除非依赖性实体所依赖的实体已经存在了，否则它是不能存在的。

依赖性约束规则是可以应用于依赖实体的参照完整性约束的一种特殊形式。一些数据库管理系统自动增强了大多数的依赖性约束，而其他的系统没有这样。

表G-30和表G-31说明了一个具有多个行项的订单的依赖性。

表G-30 订单表

订 单
订单编号
PK
ZA8845
XN7691

表G-31 行项表

行 项			
订单编号	行 号	数 量	产品代码
PK			FK
FK			
ZA8845	1	10	4260067
ZA8845	2	1	3084XC8
ZA8845	3	5	5160002
XN7691	1	2	5360A23
XN7691	2	18	3180001
XN7691	3	2	3520002

1. 递归

递归关系是一种在两个具有同样类型的实体之间的关系。

递归关系出现的频率比我们通常想象的要多得多。两个最为常用的递归关系是：

- (1) 外爆/内爆材料单；
- (2) 组织的分级。

递归关系是普通关系（即一对一、一对多、多对多）当中的一种特殊情况，以完全一样的方法建模。我们可以通过制造一个代表公司组织结构的员工实体着手，如表G-32所示。

表G-32 员工表

员 工		
员工编号	员 工 名	部门编号
PK		
00100	CODD	220
00135	KENT	220
00171	NIJSEN	220

(续)

员 工		
员工编号	员 工 名	部门编号
00190	DATE	220
00326	BOYCE	220
00529	KAGOOL	354
00558	MONGO	354
00721	STEIGLITZ	354
00843	STROHEIM	955

一个经理和他的员工之间的关系（组织结构）是一个一对多的关系，作为一个外键的经理的员工编号如表G-33所示。

表G-33 员工表

员 工			
员工编号	员 工 名	部门编号	经理员工编号
PK			FK
00100	CODD	220	00326
00135	KENT	220	00326
00171	NIJSSEN	220	00326
00190	DATE	220	00326
00326	BOYCE	220	00843
00529	KAGOOL	354	00721
00558	MONGO	354	00721
00721	STEIGLITZ	354	00843
00843	STROHEIM	955	NULL

递归关系利用了附加的完整性约束，在这个例子中需要注意以下几点。

- 经理不可能自己为自己工作。这表明了分级的最高等级必须在经理员工编号列中包含一个空值。
- 经理不能为任意他的员工工作，他也不能为他的员工的手下工作，以此类推。

材料单处理模型是一个多对多递归关系的例子，在这一关系中每个组件都用于许多子集和已完成的产品中，在这些已完成的每个产品中都包括许多组件。

下面的问题作为练习。

- 这样一个模型应该看上去是什么样子的？
- 这一模型上应该加什么限制？与前一模型上所加限制相比，它们是否不一样？

2. 在数据库设计中使用该模型

除非能够对数据库设计有所贡献，否则所有建模的工作都是没有用的。在将模型转换为物

理数据库设计的过程中，一些规范化的折中是必需的，以获得令人满意的性能。折中将包括如下内容。

- 最小化地实现关系设计。
- 适当实现分级设计。
- 最大化地实现扁平文件设计。

这一部分的意图并不是使用特定的数据库系统给出实现模型的完整引导。这部分内容在IMS数据库设计和关系数据库设计课程中都有所涉及。

3. 关系型设计

使用关系数据库管理系统实现模型的第一步是按照下面的陈述实现这一模型。

- 每个实体和关系都变成一个表。
- 将逻辑相关的实体分组放入数据库。
- 每个属性变成表中的一列。
- 为每个主键定义一个唯一的索引（为了保证行的唯一性）。
- 为了支持已知的访问路径，创建附加的索引。
- 为每个表选择这样一个索引：在这个索引下数据是聚集的，以支持最常用的访问次序。
- 完成空间计算。

为了获得可接受的性能可能需要后续修改。

G.11 决策表

决策表是用来表示不同判断的组合所导致的行为集合的一种技术，也是流程图分析的一种方法。因此，每一列都包括一个测试用例或者流程图的一个路径。

为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个字段的取值范围。

程序：FIELD-COUNT

```

Dowhile not EOF
    read record
    if FIELD_COUNTER > 7 then
        increment COUNTER_7 by 1
    else
        if FIELD_COUNTER > 3 then
            increment COUNTER_3 by 1
        else
            increment COUNTER_1 by 1
        endif
    endif
End_While
End

```

这段程序对应的决策表如表G-34所示，表中有4个测试用例，用于使用决策表测试这段程序。

表G-34 决策表

判 断				
EOF	Y	N	N	N
FIELD_COUNTER > 7	—	Y	N	N
FIELD_COUNTER > 3	—	—	Y	N
动 作				
结束程序	×			
FIELD_COUNTER_7递增1		×		
FIELD_COUNTER_3递增1			×	
FIELD_COUNTER_1递增1				×

G.12 桌面检查

桌面检查是通过一个人从头至尾走查一遍进行的人工检查缺陷的过程。标准的应用是这样的：一个人阅读程序，对照检查表进行检查并且手动输入测试数据进行检查。这同样可以应用于对需求和设计工作的检查。这项技术用于在程序编写完成或者设计完成之后对程序的质量进行评估。

G.13 等价类划分

等价划分测试是一项黑盒测试技术，将输入域划分为一些导致不同行为的输入类。

根据需求，每个输入都被分到不同的划分。使用这项技术，从每个划分中只选择一个有代表性的值进行测试。我们假设产生的结果能够代表同一个划分中所有其他值产生的结果，这就说明了此项技术的实用性和经济性。

这比简单的范围测试要复杂很多，因为一个范围根据可能导致的不同行为会被分成很多范围。考虑下面的应用程序。收入需要被分为3个等价类，因为行为（税率）根据收入值的不同而不同。

一个美国国税局的程序基于收入计算州收入税金的总量，如表G-35和表G-36所示。下面是定义等价类的一些原则。

表G-35 收入与税金百分比

收入范围	应交税金百分比
1~30 500美元	25
30 501~62 500美元	27
62 501美元及以上	38

表G-36 收入/税金测试用例

测试用例编号	测试数据值	期望结果	等价划分类
1	25 000美元	6 250美元	1 ~ 30 500美元
2	40 500美元	10 935美元	30 501 ~ 62 500美元
3	85 200美元	32 376美元	62 501美元及以上

值集。例如，“输入颜色可以是红色、蓝色、白色和黑色”。举例说来，一个测试用例可能是红色、蓝色、白色、黑色。

1. 数值输入数据

取值范围。例如，“输入数据的范围为0~100的整数”的一个测试用例可以是45（1~100的任意值）。举例说来，“输入数据的范围为0~100的实数”的一个测试用例可以是75.0（0.0至100.0之间的任意值）。

2. 数值输出数据

输出值的范围。例如，“保险金计算的输出数据的范围为0.0~100 000.00的实数”。举例说来，一个测试用例可以是15 000.00（0.0至100 000.00之间的任意值）。

3. 非数值输入数据

表或数组。例如，一个测试用例可以是任意表行中读取的字符内容。

项数。例如，“和模型相关的产品数量不超过10个”。举例说来，一个测试用例可以是5个产品（0至10之间的任意个数的产品）。

4. 非数值输出数据

表或数组。例如，对任意表格行的更新、写入及删除。

输出数。例如，“最多显示10个用户”。举例说来，一个测试用例可以是显示7个用户（0至10之间任意数量的用户）。

5. 使用等价类划分创建测试用例的步骤

(1) 定义等价类。

(2) 编写第一个测试用例，尽可能多地覆盖来自规则的有效等价类（尽管它们可能是互相排斥的）。

(3) 继续编写测试用例，直到覆盖了来自规则集的所有等价类。

(4) 为每个无效类编写一个测试用例。

下面的例子说明了这个过程。

假设需求陈述了4门小汽车的成本应该在25 000美元到38 000美元之间，小汽车类型应该是Ford、Chevy、Jeep、Honda。月付款应该低于500美元。

步骤1：定义等价类。

	等价类	
24 999美元	25 000~38 000美元	38 001美元
无效	有效	无效

步骤2~4: 创建有效和无效测试用例。

有效测试用例				
测试用例	价格	门数	车型	月付款
1	30 000美元	4	Ford	450
2	30 000美元	4	Chevy	450
3	30 000美元	4	Jeep	450
4	30 000美元	4	Honda	450
无效测试用例				
测试用例	价格	门数	车型	月付款
5				
6	24 999美元	4	Ford	450
7	30 000美元	2	Ford	450
8	30 000美元	4	Lexus	450
9	30 000美元	4	Ford	38 001

G.14 异常测试

在异常测试中, 需要确定所有的出错消息和异常处理过程, 包括触发它们的条件。每个测试用例对应一个错误条件。表G-37对于记录错误条件和异常是很有帮助的。

604

表G-37 测试用例/错误异常测试矩阵

测试用例名称	出错消息/异常	通过/失败	日 期	测试者
1				
2				
3				
4				
5				
6				
7				
8				
9				

G.15 自由形式测试

自由形式测试, 也叫做错误猜想、即兴测试或者头脑风暴, 是一种“不保险的”关于错误可能出现的地方和形式的直觉猜想, 是其他测试技术的一项附加技术。

一些测试人员对这种测试方式很拿手，因为这无须使用任何特定的测试技术。这里需要的是“嗅出”缺陷的直觉和经验。现在还没有使用这项技术的特定方法，最基本的途径是将潜在的错误或者可能发生错误的情况列举成一个清单并且根据这个清单编写测试用例。

G.16 灰盒测试

黑盒测试主要是根据规约去测试程序的功能，白盒测试主要是测试程序的逻辑路径。灰盒测试是黑盒测试和白盒测试的有机结合。测试人员要研究需求规约并且与开发人员沟通以了解系统的内部结构，这样做的目的是清除一些有歧义的需求规约，掌握程序的逻辑以设计引申的测试。举例来说，当某一特定的功能在应用程序中会被重复使用的时候，测试人员可以采用灰盒测试。如果测试人员通过与开发人员交流并理解了内部的设计和架构，很多无效的测试就可以被排除掉，因为对于这个功能只测试一遍就行了。还可以举另一个例子，当命令行语法包含7个可以以任何顺序输入的可能的参数时，如下所示：

```
Command parm1, parm2, parm3, parm4, parm5, parm6, parm7(enter)
```

理论上，测试人员将不得不进行7!（即5040次）不同的测试。如果一些参数是可选的，在这种复合情况下问题会更加复杂。如果测试人员采用灰盒测试，通过与开发人员交流，理解了内部的语法分析算法，如果每一个参数都是独立的，那么仅用7次测试就可以满足要求了。

G.17 直方图

直方图是对按照出现频率或者相对频率进行组织的测量值的一个图形描述。在表G-38中，该表格由100个某应用程序的客户/服务器终端响应时间（从按键开始到系统做出响应）组成。这可以使用性能测试工具进行测量。

表G-38 100个样本的响应时间（秒）

2	4	1	6	5	12	4	3	4	10
5	2	7	2	4	1	12	4	2	1
1	2	4	3	5	1	3	5	7	12
5	7	1	2	4	3	1	4	1	2
1	3	5	2	1	2	4	5	1	2
3	1	3	2	6	1	5	4	1	2
7	1	8	4	3	1	1	2	6	1
1	2	1	4	2	6	2	2	4	9
2	3	2	1	8	2	4	7	2	2
4	1	2	5	3	4	5	2	1	2

表G-39中的直方图展示了上述表格中的原始性能数据是如何在直方图中展示的。需要注意的是，设计规约中要求响应时间要小于3秒。从数据中我们可以明显看出这个性能需求没有达到，这里存在着性能问题。

表G-39 响应时间直方图

平均时间=3.47秒									
0	23	25	10	16	10	4	5	2	5
0 ~ 0.9	1 ~ 1.9	2 ~ 2.9	3 ~ 3.9	4 ~ 4.9	5 ~ 5.9	6 ~ 6.9	7 ~ 7.9	8 ~ 8.9	9 ~ ∞

G.18 审查

审查是同级评审最正式也是最常用的形式。审查最重要的特征就是使用一个检查表来推进错误发现的过程。当统计数据表明某种类型的错误发生得更频繁或者更少了的时候,这些检查表应该相应地进行更新。虽然审查可以用于生命周期的任何阶段,但是绝大部分常见的审查类型都是在产品设计和编码阶段进行的。

因为一般来讲审查是很透彻的,所以审查不宜过长;因此,选择评审的产品组件应该小一点比较好。产生50~100行代码的规约或者设计一般来讲是可管理的。这通常需要15分钟到一个小时的审查,而复杂的组件可能需要多达两个小时。任何情况下,超过两个小时的审查都不会很高效,应当尽量避免。

在审查的前两三天,生产者要整理好审查需要的材料并将其交给协调者进行分发。参与者应该在评审之前对材料进行研究并给出意见。

评审由生产者之外的一个参与者主持。总的来说,在生命周期的下一个阶段中扮演重要角色的人在检查过程中充当读者。举例来说,需求审查由设计人员主持,设计评审由实现人员主持,以此类推。代码审查是一个例外,由设计人员主持。审查由一个指定为团队主管或者协调者的人来进行安排和协调。

读者通览整个产品组件,使用检查表来识别错误的通常类型和标准违规。审查的主要目的在于确定一些可修改的条款,以增加该组件的易理解性、可维护性或者易用性。参与者将针对他们在之前对审查学习的过程中发现的问题进行讨论。

审查结束时,组内要做出接受或者拒绝的决定,协调者要总结出发现的所有错误和问题,并将整理好的列表分发给每个参与者。这次评审的组件的所有者(比如评审程序的设计人员、实现人员或者测试人员等)使用这个列表来对该组件进行修正。修正完成后,协调者和管理者做一次小型的评审,将问题列表作为检查表使用。随后协调者完成管理和摘要报告。摘要报告用来为后续的评审更新检查表。

G.19 JAD

JAD(Joint Application Development, 联合应用开发)是通过融洽的合作研讨会,使最终用户和开发人员都参与到系统的设计之中的一种技术。研究发现,JAD与传统的设计技术相比可以提高产品的生产率。JAD不再通过一对一的访谈来收集信息。通过将客户置于主导位置,参与者之间的沟通、协调和团队合作等都得到了提升。

JAD从逻辑上可以划分为3个阶段:客户化、会议和综合报告。无论个人在开发过程中从事

什么活动，这3个阶段都是始终存在的。每个阶段都有各自的目标。

(1) 客户化——这是JAD的一个关键阶段，主要工作是对下一阶段进行准备。参与人员是会议主持和JAD分析人员。任务包括组织团队、定义JAD任务和可交付物以及为接下来的JAD会议准备材料。

(2) 会议——这个阶段由一系列分析人员和用户合作定义系统需求和设计的促进会议组成。会议主持推动会议的进行，分析人员记录讨论结果。

(3) 综合报告——在这个最后阶段，产生正式的JAD结果。会议主持将可视的文档和其他文档归纳到一个JAD文档中。设计的结果将反馈给执行负责人。

正式的开发过程可能是由一系列以上3个阶段所组成的，直到完成最终的需求和设计。当项目有多个设计活动时（例如，总体设计的不同部分），在设计完成之后会有一个最终的综合报告，作为对设计的一次综合评审。

G.20 正交表测试

正交表测试是由Genichi Taguchi博士发明的用于制造业的统计方法，在测试用例的选择中帮助我们解决组合因素的庞大潜在数量这个问题。它计算所需测试的理想数量并且确定出输入值和条件的各种变化。举例来说，这种方法通过使用最少数量的测试用例来实现最大程度的覆盖来帮助进行测试选择。

Taguchi方法提到了质量工程学的技术，包括了统计过程控制（SPC）和新的质量相关的管理技术。对于Taguchi方法的大部分关注和讨论都集中于该过程的统计学方面，需要强调的是该方法为了质量提高和过程健壮性而设计的概念框架。

举例来说，一个命令含有3个可能的参数，每个参数有3个可选值，如下：

```
Command PARM1, PARM2, PARM3 (enter)
PARMx = 1,2,3
```

理论上，测试人员需要创建33或者27个测试组合，如表G-40所示。

表G-40 参数组合（完整列举）

测试用例	PARM1	PARM2	PARM3	测试用例	PARM1	PARM2	PARM3
1	1	1	1	10	1	2	1
2	1	1	2	11	1	2	2
3	1	1	3	12	1	2	3
4	2	1	1	13	2	2	1
5	2	1	2	—	—	—	—
6	2	1	3	14	2	2	2
7	3	1	1	15	2	2	3
8	3	1	2	16	3	2	1
9	3	1	3	17	3	2	2

(续)

测试用例	PARM1	PARM2	PARM3	测试用例	PARM1	PARM2	PARM3
18	3	2	3	23	2	3	2
19	1	3	1	24	2	3	3
20	1	3	2	25	3	3	1
21	1	3	3	26	3	3	2
22	2	3	1	27	3	3	3

应用正交表测试 (OATS), 该技术选择测试用例来测试独立测量量 (称为因素) 之间的交互作用。每个因素有一个有限的可能值的集合, 称为层次。表G-40中有3个因素 (PARM1、PARM2和PARM3), 每个因素有3个层次 (1、2和3)。这项技术要求测试人员将因素和层次的最佳数量的组合填入正交表中 (很多统计书中都有提到)。在表G-41中, 选择了3个因素和3个层次的正交表。正交表中的每一列代表一个因素, 每一行代表一个测试用例。这些行中包括了所有参数两两之间所有的层次组合。因此, 最终仅仅需要9个测试用例, 这显示了这项技术的能力。

609

表G-41 参数组合 (OATS)

$L_9(3)^3$ (正交表, 3个因素, 3个层次)			
测试用例	PARM1	PARM2	PARM3
1	1	1	3
2	1	2	2
3	1	3	1
4	2	1	2
5	2	2	1
6	2	3	3
7	3	1	1
8	3	2	3
9	3	3	2

G.21 帕累托分析法

帕累托图 (Pareto diagram) 是一个特殊格式的图, 它指出了应将精力集中于哪个部分。按照频率 (或者花销、失败率等) 的递减顺序来描述事件或者事实, 这能够帮助我们迅速地从很多琐细的部分中找到那几个重要的。信息系统中, 帕累托图的80/20规则更加广为人知, 就是说, 20%的原因导致了80%的出现频率。帕累托图是一个直方图, 以递减顺序显示数据, 帮助我们找到问题最高频率的起因, 以采取适当的处理方法。这也是一种根据原因的类型对问题起因的一个排名, 目的在于找到导致问题的频率最高的一个或几个起因, 定位消除这些起因的行为。

下面是使用帕累托图的4个步骤。

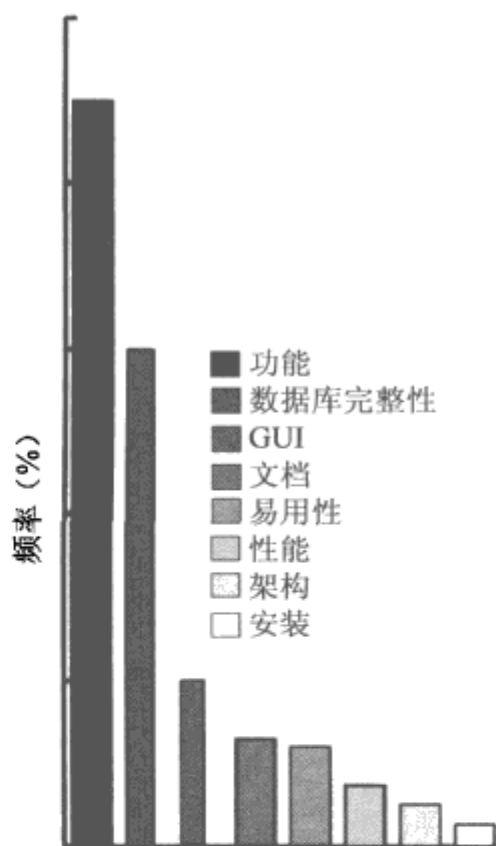
(1) 确定问题区域。一个问题的例子是软件测试阶段发现了过多的缺陷。

(2) 确定并且标识问题的起因。这是最耗时的一个步骤，因为它需要从不同的起因中收集信息。缺陷的起因包括架构、数据库完整性、文档、功能、图形用户界面、安装、性能和易用性等。对于大部分问题，很少需要定义超过12个起因。当标识出超过12个起因时，一种方法是选取其中的11个并将第12个分类为“其他”。如果发现“其他”占有了很大的比例，则需要将其分解为具体的起因。

(3) 记录问题起因的每次出现。起因的出现次数需要记录下来。缺陷跟踪数据库中的样本可以用于获得需要的频率。

(4) 使用帕累托图，按照频率对起因进行排名。这包括两个任务，一是按类型对问题出现次数进行记录，二是建立一个直方图（帕累托图），将最主要的起因列在最左边，其他起因按频率的降序从左至右排列。

在图G-2中有8个缺陷起因，记录了大约1050个缺陷。在这当中，750个由功能和数据库完整性引起。也就是说，20%的起因导致了71%（接近80%）的缺陷比率。在这个例子中，功能是最主要的起因，其次是数据库完整性，所以主要精力应放在消除功能和数据库问题上。一种解决办法是增加单元测试和评审。



图G-2 帕累托图

G.22 正反测试

正反测试是一种基于输入的测试技术，它需要进行适当比例的正面测试和反面测试。正面测

试是具有一个有效输入的测试，相应的反面测试则是具有一个无效输入的测试。因为一般来说反面测试要比正面测试多得多，因此建议使用的比例是80%的反面测试和20%的正面测试。

例如，假设一个应用程序接受5个字符变量的股票市场或者共同基金，然后显示相应的股票或共同基金名称。一个正面测试的例子是“PHSTX”，这是与一个健康科学基金相关的共同基金标识。如果这个标识显示了一些其他的基金，这就是一个失败的正面测试。

无效的股票或共同基金的标识值是反面测试。一般来说，反面测试产生一个无效出错消息。例如，如果输入“ABCDE”，然后显示了一条无效出错消息，这就是一个通过的反面测试。

对于反面测试需要考虑的问题是：需要多少反面测试才是足够的？以及我们如何预知一些非期望的情况。测试一个单个字母字符的字段的编辑可能是非常复杂的。它的一个反面测试可能是“(”，而且系统应该能够测试出它。“)”也要测试吗？应当测试多少其他的非字母字符？有些时候预计外的条件也是非常难测试出来的。例如，“&”和“'”在SQL中是具有特殊意义的。这两个字符在每个字段中都要测试吗？

G.23 缺陷历史预测测试

在缺陷历史测试中，为每一个在系统以前的测试中发现的缺陷都创建或重运行一个测试用例。这样做的原因是缺陷一般会集聚在一起并回溯到原始问题。导致问题的一些原因包括糟糕的软件配置管理规程、糟糕的代码编写和在缺陷修复过程中的单元测试、错误有集聚在一起的倾向性等。

缺陷矩阵是一个将测试用例与功能（或者程序单元）相关联的优秀工具。缺陷矩阵中的一个检查项表明需要重测试该测试用例，因为在之前运行这个测试用例的时候发现了一个缺陷。空白项说明这个测试用例不需要重新运行。

如果因为发现了许多的缺陷而使这个方法变得不够经济，应当选择某一重要性等级以上的测试用例进行重测试。

G.24 原型法

原型法是一种迭代方法，用于建立用户在初始时不能精确描述的系统。它的概念很大程度上是通过第四代语言 and 应用程序生成器的作用实现的。原型法是同任何其他开发工作一样易于产生缺陷的，如果没有将其用系统化的方法运行的话，也许还会产生更多缺陷。原型需要像任何其他系统一样进行彻底的测试。如果没有为开发原型建立一个系统化过程的话，原型测试将是非常困难的。

下面几节会描述几种原型法。这几种方法说明了用于定义软件生命周期的概念的多样性，并论述了原型法在总体上对生命周期产生的影响。

G.24.1 循环模型

这一使用原型法的软件开发的观念由两个单独的但是内在有联系的循环模型组成，其中之一由一个传统的软件开发周期组成，另一个则由一个原型周期组成，这一原型周期在分析与设计阶

段与该传统模型相互作用。主要的操作如下所述。

- 传统周期：
 - 用户请求；
 - 可行性；
 - 调查；
 - 考虑原型；
 - 分析；
 - 设计；
 - 最终提交设计；
 - 编程；
 - 测试；
 - 实现；
 - 操作；
 - 评估；
 - 维护；
 - （重复这一周期。）
- 原型周期：
 - 设计原型；
 - 使用原型；
 - 使用原型进行调查；
 - 在调查基础上进行分析；
 - 进行精炼或者建立一个新的原型；
 - （重复这一周期。）

两个周期的相互作用发生在传统周期中的调查发现需要使用原型法的时候，这时进入原型周期。当传统周期的分析、设计或最终提交设计能够在原型周期中发现或验证的信息的基础上完成的时候，终结原型周期。

613

G.24.2 第四代语言和原型法

这一方法给出了以下生命周期的步骤。

- (1) 组成一个原型法的队伍，包括一个分析人员/程序员和一个终端用户。
- (2) 通过访问一些终端用户标定用户需求以定义问题并对用户期望采样。
- (3) 很快地开发出一个原型以阐述问题和用户期望的大部分议题。
- (4) 将原型演示给终端用户。用户使用原型进行体验并在一个特定时间周期内进行工作。如果原型是无法接受的，丢弃它。
- (5) 通过将在使用中标记的变动合并进去，可以对原型加以精炼。这一步骤和前一步骤是循环进行的，直到系统最终达到需求标准。
- (6) 组成一个终端用户小组以便在一个特定时间段内对原型给出更多反馈。

(7) 决定是否要实现该原型或者系统是否需要用常规语言重写。做出这个决定要基于可维护性、硬件和软件效率、灵活性和其他系统需求。

G.24.3 迭代开发对账

这个模型是基于如下观点的：系统是一系列的规约层次，每一层都比上一层更加详细。这些层次包括：

- 非正式需求；
- 正式需求；
- 设计；
- 实现；
- 配置；
- 运转。

每一层都比上一层有更多的详细内容。另外，每一层都要与上一层的规约相一致。迭代开发需要对每一层都进行开发“对账”（也就是说，如果某个规约层次想进行一个变更，首先更高层次要做出改动来支持这个变更）。

开发的完整历史有这个“对账”记录来维护，保证所有层次上的一致性。每个层次上都建立一个原型来说明规约是一致的。每个原型都专注于在该层次上实现的功能。当测试、安装和培训结束后，最后的原型就成为了实现的系统。

614

G.24.4 演化和舍弃

这里是两种模型。第一种模型中，原型建立并且不断增强，最终形成实现的系统。另外一种模型则是舍弃模型。

在这两种模型中，终端用户都是原型开发的完整部分，应当学习如何使用原型构建工具（举例来说，一种仿真语言或者4GL）。这两种模型的简要描述如下。

- 方法1
 - 用户体验并且使用为响应终端用户最早也是最希望的对系统工作方式的要求而建立的原型。
 - 分析人员观察用户行为，确定原型的哪个部分需要精炼。一系列的原型，或者说对初始原型的更改，演化成了最终产品。
- 方法2
 - 已经实现了一个原型。从这个原型和终端用户的反馈得出初始设计。建立另外一个原型来实现初始设计。用一种常规语言实现最终系统。

G.24.5 应用程序原型法

这一方法包括以下步骤。

(1) 基本需求的确定——把精力集中在确定基本目标和需要解决的主要业务问题，并定义数

据元素、数据关系和功能。

(2) 工作模型的开发——迅速建立一个工作原型，以表述关键需求。

(3) 原型的演示——将原型向所有有兴趣的用户展示，并通过用户反馈获得附加的需求。

(4) 完成原型——不断重复演示和模型增强这两步，直到用户对组织能够提供原型所需的服务满意。一旦用户认同原型满足服务所需的概念，则可将原型增强成为最终系统或者用一种更有效率的语言重新编写。

G.24.6 原型系统开发

这一方法的各阶段如下所述。

(1) 管理层阐述组织的目标。这是以信息需求和系统边界和容量的范围的形式描述的。开发出原型的界面和报表。

(2) 终端用户和管理层对原型加以评审和批准。完成完整的系统设计、设备选择、编程和文档化。

(3) 管理层评审并提交实现的系统。与旧系统平行运行原型的系统测试。开始下一版本的工作，这就造成了以上3个阶段的循环进行。

G.24.7 数据驱动原型法

原型法是一种理清设计思路、测试假设和从用户处收集实时反馈的很好的沟通工具。

这一方法由以下步骤组成。

(1) 操作检查——定义项目范围并评估环境、当前组织和信息结构。

(2) 概念设计——定义给出的元数据（也就是数据的结构以及在两个个体结构之间的关系）、描述改变数据状态的服务功能所需的场景和检索的类型。

(3) 数据设计——规范化元数据。

(4) 启发式分析——在真实数据值的使用过程中，对照元数据检查需求的一致性，这一步骤与数据设计步骤一同循环。

(5) 环境测试——构建程序以支持数据输入和检索（原型）。

G.24.8 传统生命周期的替代

在这种模型中，包括如下步骤。

(1) 快速分析——导致一个不完整的纸上模型，它显示了系统的上下文、关键功能、一个数据库的实体关系模型和概念的表格、界面、属性、报告和菜单等。

(2) 数据库开发——使用一个关系架构来为使用原型创建一个工作的数据库。

(3) 菜单开发——详述在快速分析中定义的初始概念并修补应用程序的层次结构。

(4) 功能开发——将功能按照类型分类到模块中。

(5) 原型展示——通过在必须时重复做上面的部分工作并在必要时进行调整以循环迭代。

(6) 设计、编码和测试——完成详细的设计规约。

(7) 实现——基于原型的演化和所有程序测试和文档的完成。

G.24.9 早期原型法

这种模型可以帮助细化用户需求、验证系统设计的灵活性，并将原型转换成最终系统。这一过程包括如下内容。

- (1) 预先分析和需求规约为将来再引用时建立一个基线。
- (2) 定义并实现一个原型，强调用户界面。一个小开发团队使用原型开发语言和辅助快速开发的工具开发出原型。
- (3) 在用户的工作空间中测试这一原型。
- (4) 尽快通过采纳用户意见来精炼原型。
- (5) 通过采纳从原型中获取的经验精炼基线需求。
- (6) 通过使用一个具有源于原型的需求的传统生命周期开发出产品系统。

G.24.10 用户软件工程

这基于一种软件开发的模型，该模型是部分正式而部分非正式的，并包括如下步骤。

- (1) 需求分析——活动和数据建模以及用户特性的标定。
- (2) 外部设计——开发事务和用户程序界面。
- (3) 门面开发——用做一个用户程序界面的原型，并在需要时加以修正。
- (4) 叙述文本——用于非正式地详述系统操作。
- (5) 初步关系数据库——作为系统的一个功能性原型的基础而设计。
- (6) 功能性原型——开发出来用于提供给定系统的一些或者全部功能。
- (7) 系统操作的正式规约——在此时是可选的。
- (8) 系统体系结构和模块——概念设计并定义整个结构和相关软件模型。
- (9) 系统实现——使用一种过程语言。
- (10) 测试和验证——在系统发布到实际应用环境中之前在系统上完成。

617

G.25 随机测试

随机测试是这样一种技术，其中程序或系统通过在所有可能输入值中选择一个随机子集的方法进行测试。它不是一种最优的测试技术，因为它测试到很多缺陷的几率很小。但是有时它能发现一些标准化的测试技术不能发现的缺陷，因此它被作为一个附加的测试技术。

G.26 范围测试

范围测试是这样一种技术，它假设在一个预先定义的范围中的任意输入变量的行为都是一样的。首先应当选择系统行为完全一样的范围，然后选择并测试所选范围中的任意一个代表，如果它通过了，则假设其他的值不需要再被测试。

举例来说，考虑以下代码段，它计算了两个输入值 X 和 Y 给出的结果 Z ：

$$Z = \sqrt{X^2 - Y^2}$$

如果 X 和 Y 是0~5的正整数并且 X 大于等于 Y ，则有21个可能的测试用例，如表G-42中所示。

表G-42 范围测试的测试用例

测试用例	X 值	Y 值	Z (预期结果)
1	0	0	0
2	1	0	1
3	1	1	0
4	2	0	2
5	2	1	$\sqrt{3}$
6	2	2	0
7	3	0	3
8	3	1	$\sqrt{8}$
9	3	2	$\sqrt{5}$
10	3	3	0
11	4	0	4
12	4	1	$\sqrt{15}$
13	4	2	$\sqrt{12}$
14	4	3	$\sqrt{7}$
15	4	4	0
16	5	0	5
17	5	1	$\sqrt{24}$
18	5	2	$\sqrt{21}$
19	5	3	4
20	5	4	3
21	5	5	0

使用这一技术，将有可能节省许多生成测试的时间，但是它具有一个限制，即假设选择任意一个输入样本将产生与其他输入相同的系统行为。如条件 X 和 Y 为正整数且 Y 大于等于 X 这样的附加测试也应该做。同样，平方根结果的验证也应该被测试。例如，我们需要确定是否 Z 变量可接受分数值作为计算或中断的结果（参见G4节）。

G.27 回归测试

回归测试根据在一个开发螺旋周期中调试、维护或开发一个新版本的产生的变化对应用程序加以测试。这一测试必须在功能改进或系统修补已经完成之后进行，以确定这些改动没有造成意料之外的负面影响。与逻辑和控制流相关的错误、计算错误和接口错误的修正是需要进行回归测试的条件。装饰性错误一般不影响其他功能，也就不需要进行回归测试。

最理想的情况是测试集中的所有测试在新的螺旋中都重新运行一遍，但是由于时间限制，这经常是不现实的。在螺旋式开发中，一个好的回归策略是对于一些回归测试要在每个螺旋中都运行以保证在以后的开发螺旋或错误更正中不会对以前展示过的功能产生负面影响。在系统已经稳定并且功能都被见证过之后的系统测试期间，回归测试应当由系统测试的一个子集构成。为了决定要包括哪些测试，应当建立一些策略。

理论上来说，一个被修改过的系统的可靠性在没有对所有测试进行完整的回归测试之前是不能保证的。但是有很多实际的问题需要考虑。

- 当发现缺陷时，需要创建附加的回归测试。
- 应该有一个可用的回归测试库，并随着其发展演化对其加以维护。
- 应该有一种分离出聚焦于某一特定区域的回归测试的方法（参见重新测试和缺陷矩阵）。
- 如果一个系统的整个体系结构改变了，需要进行完整的回归测试。
- 强烈推荐使用带有捕获/回放特性的自动化测试（参见本书第六部分）。

G.28 基于风险的测试

风险管理测试的目的是，测量应用程序系统所具有的业务风险的程度以对测试加以改进。这是通过两种方法完成的：可以标识出高风险的应用程序并对其进行更高强度的测试，并且风险分析有助于标识出个体应用程序易于出错的部分，这样可以使测试能够指向这些部分。

风险测试是标识出脆弱区域的正式方法（也就是会造成潜在损失的区域）。任何可能被有意或无意的误用并导致损失的区域都是一个脆弱区域。风险标定使得测量这些脆弱区域的潜在影响的过程成为可能（例如，风险或脆弱区域可能造成的最大损失）。

基于风险的测试是一种为以前标识过的每一个主要的风险因素创建测试用例的技术。要测试每种情况以保证风险被消除了。

G.29 运行图

运行图是关于质量特性怎样随时间变化的图形表示。通常情况下它是一个线图，展示了在度量过程或者项目计数中的变化。例如，在表G-43中，运行图可以给出随着时间检测出的缺陷数目的变化。它可以给出具体种群样本或者百分比中得到的结果。

控制图，也就是运行图的特殊形式，在图中画线以表示所允许的变化的限制。这些限制可以通过设计规约或者一致认同的标准来确定。频繁设定控制限制是为了给出可能由于偶然事件造成的变化的数值限制。这可以通过使用平均数和每个样本数据的量度范围计算得到。控制图不仅仅可以用做超出限制的一个警告，也可以用于检查在限制范围之内发生的趋势。例如，如果表G-43中的10个量度的次序显示出落在预期平均值以下，则可以假设这不是仅仅由于偶然造成的并且应当在以后进行一个调查。

表G-43 运行图样本

x (周)	y
1	10
2	50
3	30
4	60
5	25
6	50
7	75
8	45

G.30 三明治测试

三明治测试同时使用自顶向下和自底向上技术并且是对两种技术做出了折中。这一方法同时结合了自顶向下和自底向上的方法，在分层控制结构的中间相遇。中间的结合点是由程序结构定义的。

这一方法一般用于大型程序但是很难证明它也适用于小型程序。层次结构的最高等级通常包括系统的用户界面，需要桩来模仿业务功能。最底层包括需要驱动器以模仿更低级别模块的原始级别模块。

621

G.31 语句覆盖测试

语句覆盖是一种白盒测试技术，确保代码的每一条语句或者代码行都至少执行一次。它确实保证了每条语句都被执行，但是这是一个非常弱的代码覆盖技术并且不像其他技术那样完善，比如分支覆盖，其中判断语句的分支都被执行。

为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个字段的取值范围。

程序：FIELD-COUNT

```
Dowhile not EOF
  read record
  if FIELD_COUNTER > 7 then
    increment COUNTER_7 by 1
  else
    if FIELD_COUNTER>3 then
      increment COUNTER_3 by 1
    else
      increment COUNTER_1 by 1
    endif
  endif
End_While
End
```

满足语句覆盖的测试用例如下所示：

测试用例	值(FIELD_COUNTER)
1	>7, 比如8
2	>3, 比如4
3	≤3, 比如3

G.32 状态转换测试

状态转换测试是这样一种测试技术，它首先标识一个系统的状态，然后编写一个测试用例来测试造成从一个条件转换到另外一种状态的触发条件。可以使用有限状态图或者等价表

格设计测试。

为了举例说明这项技术，我们来看下面这个小程序，该程序从一个文件中读取记录并记录下每条记录中的一个字段的取值范围。

程序：FIELD-COUNT

```

Dowhile not EOF
  read record
  if FIELD_COUNTER > 7 then
    increment COUNTER_7 by 1
  else
    if FIELD_COUNTER>3 then
      increment COUNTER_3 by 1
    else
      increment COUNTER_1 by 1
    endif
  endif
End_While
End

```

表G-44论述了使用这一测试技术产生测试用例。状态被定义为COUNTER_7、COUNTER_3和COUNTER_1的当前值，然后考虑可能的转换，它们由文件结束条件或者每个成功记录输入的FIELD_COUNTER值组成。对于每个转换，要定义如何形成相关状态。每个转换都成为一个测试用例，最终状态就是预期的结果。

表G-44 状态转换表

初始状态	测试用例（转换）	最终状态
COUNTER_7=X1	1. EOF	COUNTER_7=X1
COUNTER_3=X2		COUNTER_3=X2
COUNTER_1=X3		COUNTER_1=X3
		退出程序
COUNTER_7=X1	2. 下一条记录的FIELD_COUNTER>7	COUNTER_7= (X1+1)
COUNTER_3=X2		COUNTER_3=X2
COUNTER_1=X3		COUNTER_1=X3
		成功
COUNTER_7=X1	3. 下一条记录的 $7 \geq \text{FIELD_COUNTER} > 3$	COUNTER_7=X1
COUNTER_3=X2		COUNTER_3= (X2+1)
COUNTER_1=X3		COUNTER_1=X3
		成功
COUNTER_7=X1	4. 下一条记录的FIELD_COUNTER<=3	COUNTER_7=X1
COUNTER_3=X2		COUNTER_3=X2
COUNTER_1=X3		COUNTER_1= (X3+1)
		成功

G.33 统计概况测试

在统计概况测试中,使用统计技术来开发系统的使用概况。基于期望的使用频率,测试人员确定有利于测试的事务路径、条件、功能区域和数据表格。因此,这些测试都是面向系统中使用率最高的部分的。

G.34 结构化走查

结构化走查比代码阅读评审更加正式。评审开始前要首先分配好不同的角色和职责。准备工作更多,而且强调要有记录问题的正式方法。这种评审的另一个关键属性就是它是由模块的生产人员来主持的。最常见的走查发生在设计和编码阶段;然而,近来也应用到了规约文档化和测试结果中。

生产人员安排评审的时间并且对输入进行整合和分配。大部分情况下,生产人员选择走查的参与者(虽然有时由管理层负责)并且通知他们自己所扮演的角色和职责。一般来讲,走查应不到7个人参与且耗时不超过2个小时。如果需要更多时间的话,应当给予适当的休息或者减小产品的规模。走查中的角色包括生产人员、协调者、记录人员以及用户、维护人员和标准组织的代表等。

虽然评审是由协调者开始的,生产人员有带领整个团队走查这个产品的责任。在设计和代码走查的情况中,生产人员要模拟组件的操作,允许每个参与者根据各自规约的范围给出意见。所有的问题保存到一个清单中,在评审的最后,每个参与者都要在清单上签字以表明对产品的态度:接受现在的产品,接受进行了推荐的修改之后的产品,或者不接受该产品。生产人员根据自己的判断来考虑是否接受修改建议。完全接受评审意见没有任何形式意义。如果在产品的整个生命周期中都要进行这种走查,那么上一次评审的意见在下一个评审开始时应当被讨论。

G.35 语法测试

语法测试是语法命令生成器根据系统的语法规则来生成测试用例的一种测试技术,包括创建合法和非法的两种值。这种方法是一种数据驱动的黑盒测试技术,测试输入到语言处理器(如字符串处理器和编译器)的输入数据。测试用例是基于严格的数据定义开发的。合法输入是用BNF表示法进行描述的。

语法测试的主要优点是可以确保合法数据和非法数据之间没有歧义,并且使用这种技术时会出现规约方面的问题。

G.36 表测试

表测试是这样一种技术,它测试通常与关系数据库相关联的表(同样的方法可以用于数组、队列、堆的测试)。通常有两种形式的表:顺序表和索引表。下面是对表进行的一些常规测试。

(1) 索引表

- a. 删除表中的第一条记录。
- b. 删除表中的一条中间记录。
- c. 删除表中的最后一条记录。
- d. 添加一条记录到表的第一个位置。
- e. 添加一条记录到表的中间位置。
- f. 添加一条记录到表的最后一个位置。
- g. 尝试添加一条重复记录。
- h. 添加一条带有无效键的记录, 例如, 在键字段内填入垃圾数据。
- i. 改变一条已存在记录的键, 例如, 改变一个订单号。
- j. 删除一条不存在的记录, 例如, 输入的要删除的键和表中实体不匹配。
- k. 更新并重写一条已存在的记录。

625

(2) 顺序表

- a. 尝试从一个空表中删除记录。
- b. 从一个空表中读取记录。
- c. 向一个满表中添加一条记录。
- d. 从只有一条记录的表中删除一条记录。
- e. 读取最后一条记录。
- f. 读取最后一条记录的下一条记录。
- g. 在表中顺序滚动。
- h. 插入一条序列之外的记录。
- i. 尝试插入一条重复记录。

G.37 线序测试

线序测试是通过测试应用程序中一系列的完成某个业务的程序单元, 来确保关键功能能力的一种软件测试技术。

基本上, 一条线就是一系列的功能组成的一个业务事务。把整个系统线序化, 分离成一些单独的处理过程。每个功能单独测试, 然后一次一个地增加到这条线中。接着就是测试这些业务事务线索。把测试过的线索当作子系统按照顺序集成和增量地进行测试, 这样整个系统也就测试完成了。这种方法可以使早期系统测试和验收测试更加容易。

G.38 自顶向下测试

自顶向下测试技术是一种增量式的测试方法, 它在第一时间对高层模块或系统组件进行集成和测试, 然后按照层次逐步进行测试, 直到处于底层的模块或组件。这种技术要求建立一些桩模块。当一个模块或系统组件被测试的时候, 它所调用的模块或组件由桩模块来代表, 这些桩模块连同模拟出来的结果一起把控制返回调用它们的模块或系统组件。当测试过程沿着程序结构向下进行的时候, 每一个桩模块会用它所代表的实际代码来代替。模块的先后测试顺序没有一定之

规，但唯一的规则是调用其他模块的模块或系统组件必须要先进行测试。

自顶向下测试可以较早地发现程序顶层的一些主要的设计缺陷，因为顶层的一些功能被测试的早，而这些功能又处在控制结构的顶层，这样就实现了对程序设计的早期验证。原型或初步设计越早，就会越早实现演示版本。因为菜单通常处于控制结构的顶层，所以外部界面可以尽早地展现给用户。桩模块需要建立，但通常来说比驱动模块的创建要容易一些。从另一方面来讲，一些很重要的底层模块或系统组件直到后期才能够被测试，但发生在这些底层模块的问题一般不会导致系统的重新设计。

626

G.39 白盒测试

在白盒测试（又称结构测试）中，测试条件主要是针对检查逻辑路径来设计的。测试人员检查程序或系统的内部结构。测试数据根据对程序或系统的逻辑检查来驱动，而不去关心程序或系统的需求。测试人员知道程序的内部结构和处理逻辑，就像汽车维修工知道汽车的内部构造一样。这一类测试包括基本路径分析、语句覆盖、分支覆盖、条件覆盖、分支/条件覆盖等。

白盒测试的一个优点是测试比较彻底，并且侧重于已经开发出来的代码。因为测试人员知道内部结构或逻辑，一些致命的错误或程序员故意放置一段代码而搞的恶作剧就会很容易被检查出来。

白盒测试的一个缺点是不能验证规约的正确性，也就是说，白盒测试仅仅侧重于测试内部的处理逻辑，而不去验证逻辑是否满足需求规约。白盒测试的另一个缺点是无法检验代码中遗漏的路径和数据敏感性错误。例如，如果程序中的代码语句应该写成“if $|a-b| < 10$ ”，但却写成了“if $(a-b) < 1$ ”，没有详细的规约的话，这种错误将无法被检测出来。白盒测试的最后一个缺点是无法穷举程序中所有可能的逻辑路径，因为穷举导致的测试数量将会是一个天文数字。

627

术 语 表

ANSI American National Standard Institute的首字母缩写，为广大行业创建标准的组织，包括计算机编程语言。

ASCII 用于信息交换的美国标准编码，是几乎所有计算机和打印机使用的标准化编码系统。

COE 凭借卓越中心IT组织通过集中某些或全部与测试相关的活动改进其测试实践。

CPU 中央处理器，计算机的大脑。

CRUD 创建、阅读、更新和删除。

GIGO 代表“进来垃圾，出去垃圾”。计算机与人不同，它可以毫无问题地处理没有意义的输入数据并产生没有意义的输出。

GUI 图形用户界面（Graphical user interface）——屏幕上由图形及文字构成的用户界面，用来与用户进行沟通。

ISO9000 由国际标准化组织（International Standards Organization, ISO）在1987年制定的质量系列，包含一组五个文档。

PDCA 计划、执行、检查和改进。

PMBOK 项目管理协会的项目管理知识体系。

ROI 投资回报率。

SMC 简单、中等和复杂测试用例。

SOA测试 观察整个业务流程，并确保业务流程的各个部分都正确交互。

安全性测试 安全性的基石是保密性、完整性和可用性。

按钮 显示在计算机屏幕上的，是机器按钮的可视化等价物。

版本管理 无紧急修正的、已完成且可适应项目的正式发布过程。

备份 用以进行恢复的创建文件副本的过程。

必需的需求 与不是必需相对的实际上必需的需求。

标准 用以评估产品/程序和确定不一致性的尺度。

捕获/回放 自动回归测试工具，记录和回放软件功能以验证软件编程没有反过来影响已经过测试的应用程序的任何部分。

捕获/回放测试 使用捕获/回放工具记录交互场景的测试。

测试成熟度 当前过程相对于过程标准集的差距。

测试覆盖率 在测试过程中，对实际已测试的系统某一部分的度量。

测试工具 用来测试系统的手动/自动的程序或软件。

测试评估 考虑为了完成计划的测试而要求的资源的类型和成本。

测试事件 描述测试级别的常用术语。例如，单元测试、集成测试和系统测试。

测试数据生成器 创建数据的测试工具，这些数据然后被自动化测试工具读取并输入到应用程序中。

测试套件 测试用例的集合，专门用作软件程序的输入，表明程序有某些专门的行为集合。

测试周期 用于测试系统符合逻辑且已完成部分的一组有序的测试条件。

测试准备工作评审 主要是指在一个正式测试事件开始前，确保所有准备工作和开始标准都已准备妥当并核实的正式评审。

层叠 应用程序的一条命令，可将屏幕上所有的窗口自动进行整齐排列。

窗口 屏幕上展示信息的方形。

单元测试 对单个程序进行测试以确定其运行满足需求。

动态测试 通过执行一个或多个测试来测试程序或系统。

范围陈述 包括项目资源的成本的早期评估。

非冗余的需求 不应存在重复的需求，因为这可能会引发问题。

分布图 显示两个因素间是否存在关联的图形。

负载测试 通过模拟多个用户同时对应用软件的期望行为建模的一种实践。

根本原因分析 是基于数量资料的有序过程，用以确定将缺陷引入产品的主要原因。不仅要修复对产品的影响，还要证实这个过程或方法是如何将缺陷引入产品中的。

关键字驱动的框架 不同的屏幕，功能和业务组件被指定为数据表中的关键字。

管理 管理资源的团队或个体。

管理评审及批准 管理检查是对提交内容的最后评审。由项目经理和项目发起者共同进行，以保证产品业务各方面的质量。

规程 采取按部就班的方法以实现某种标准。

过程 为实现某一功能而执行的特殊活动。

过程改进 为更快、更经济、或是更优质地生产产品而对过程进行的变更。

合规性测试 判断特定的实现规约的产品实现是否完成了指定的所有强制性元素，这些元素是否可操作。

回归测试 对再次修改后的已测试系统进行验证时所使用的测试。

混合测试框架 它是由核心数据引擎、通用组件函数和函数库定义的。

基线 (1) 特殊产品中定义好的一组执行或文件，在对所有开发和变更活动进行严格管理的情况下，在设定时间内支持某一已定义的活动。例如，集成测试、试验、系统测试、评审；(2) 经过正式评审、批准并达成一致意见的产品、文档或交付内容；之后作为以后开发的基本原则保存起来，任何变更需要经过正式的变更控制流程之后才能得以执行。例如：产品的最初部署；现有产品的演化。

基线测量 带有特殊目的，即确定某种状态初始值的测量。

基准 用以衡量硬件或软件产品相关性能的测试。

即兴测试 没有正式的测试用例（即考验和错误）的测试。

集成测试 (1) 把个体与单元测试代码段相结合的测试, 以成为一个完整单元; (2) 由时间周期引发的测试事件, 在测试开始前确定。这一测试阶段是用以确定软件产品的功能问题, 是进行验证工作。

价值 对个体有正面影响的观念或惯例。

架构/体系结构 与建筑物的体系结构相似, 计算机的体系结构是指计算机的设计结构及其所有细节。

假设 能够减少管理问题相关变量的建议。

简洁的需求 一个好的需求必须杜绝出现冗词赘句或多余信息。

结束标准 对数量与质量的测量, 用以评估特定开发阶段或状态的产品的可接受性。

紧急修复 需要立即进行的软件修复工作。

进入标准 对数量与质量的测量, 用以评估进入下一开发状态或阶段的产品的准备工作。

静态测试 通过评审过程测试制品。

可测试的需求 可测试的需求必须可以被验证或确认, 也就是说, 需求的意图应该是能够证明的。

可跟踪的需求 需求必须是可跟踪的。可跟踪性是验证是否满足需求的关键。

可理解的需求 可理解的需求是用便于评审的方式组织的。

可修改的需求 需求及相关信息必须是可修改的。

客户 接收产品的个人或组织。

客户端/服务器架构 一种系统架构, 客户端所在计算机通过网络与服务器进行合作。

控制图 用以区分过程中所展示的普通原因和特殊原因变化的一种统计学方法。

流程图 展示过程的步骤顺序的图表。

敏捷方法论 将迭代开发、测试和反馈结合在一起的一个值、原理和实践的集合。

模块化框架 要创建小的独立自动化脚本和代表待测应用程序的模块、部分、函数的功能的一种方法。

平均值 将各个项目的和除以项目个数得出的值。

评审 在处理过程或会议中让项目人员、项目和程序经理、使用者、客户、发起人或其他感兴趣的第三方为一个或一组工作产品提出意见或建议。

缺陷 从生产者(制造者)角度——没有满足产品需求。从客户角度——所有使客户不满意的地方。

确认 一种评估类型, 用以在开发阶段后期评估软件产品是否满足需求。

软件维护 在应用程序投产后发生的所有变更、修正及改进。

容量测试 性能测试的一种形式, 在这种测试中将数据增加到不正常的量, 以观察系统的行为。

生产率 使用相同测量单元的某一过程的输出与输入比率。

事故报告 以文档形式记录测试执行过程中产生的问题或错误的报告。

适应性维护 对系统进行修改以适应操作环境中的变更。

输入 过程中所需的产品、服务、或者信息。

属性 描述的事物特征。

数据库 采用计算机格式存储的信息集合。

- 数据驱动测试** 一种框架,在这种框架中测试输入和输出值被从数据文件中读出。
- 算法** 认为可以为特殊问题给出正确答案的一组规则。
- 探索性测试** 由挑战假设驱动的软件故障和缺陷战术追击。
- 统计过程控制** 使用统计学及工具测量进程。
- 图标** 代表某一功能的小图片。
- 拖放** 使用鼠标将一个图标拖到另外一个图标上的动作。
- 完善性维护** 有关软件性能、可维护性、或易理解性的提高。
- 网络** 将计算机连接起来、共享资源的系统。
- 维护** 与生产软件的修改或完善相关的任务。
- 问题** 任何与预先确定标准的偏差。
- 问题报告** 关于在软件产品、交付内容或开发过程中所发现问题的确定、跟踪及分配方法。
- 误差** 实际值或条件与其期望值之间的差异。
- 系统测试** 对系统进行的功能测试,已确保其在系统需求范围内工作并满足使用。
- 下载** 从其他电脑接受信息,通常指文件。
- 项目规章** 生动的业务文档,记录官方认可的项目投资。
- 项目框架** 将质量过程与项目各个阶段结合在一起,并将项目质量管理与系统或软件开发方法同步的一种有用的方法。
- 项目管理** 应用各种知识、技能、工具和技术满足项目需求。
- 性能测试** 从不同的角度进行测量,以改进应用程序的可扩展性和性能。
- 修正活动** 在软件产品和开发方法中的报告、跟踪及解决所发现问题的手段和过程。为所发现的问题提供了最终解决方案。
- 修正维护** 鉴定及移除代码缺陷。
- 需求** 某一属性或功能的性能标准,或者验证达到标准的过程。
- 压力测试** 增加加在系统上的负载,使其超出正常预期的用途,以测试应用的响应。
- 验收测试** 向用户确保系统正按用户期望的那样工作的一种测试形式。
- 验证** 一种用于确定软件产品在特定的开发阶段是否满足某种强制条件的评估类型。这个强制条件在此特定开发阶段的初期就已确定。
- 遗留系统** 有关产品的早期应用系统。
- 易用性测试** 在满足有效性、高效性和用户满意度的条件下,一个产品能被任何特殊用户用于达到某种特殊目的的程度。
- 因果图** 描述一些结果及其潜在起因的关系,确定问题可能起因的工具。
- 影响分析** 确定软件或硬件变更对系统哪些组件造成影响的过程。
- 用户** 使用产品或程序的客户。
- 用户故事** 用简单的语句形式对需求进行的信息陈述,通常写在3×5英寸的卡片上。
- 用例** 描述参与者使用系统完成工作的场景。
- 远景** 对某种事务期望前景的描述。
- 运行图** 按时间顺序排列的数据点所组成的图形,用以检测被测量特征的趋势。

正式评审 通常安排在开发过程每个活动或阶段结束时的一种评审类型，用以评审包含于提交内容或一些用例中的、一个完整提交内容或软件产品及其支持文档中的组件。

政策 有关于过程或产品的管理意图或目标。

直方图 对由发生频率构成的测量值的图形描述。

质量 关于符合规定或隐含需求的总体功能及产品或服务的特征。

质量标准 为每个项目计划质量管理方法，包括建立质量标准。

质量保证 需要计划和系统级操作的一般过程，以保证产品或服务达到了既定需求。

质量保证评估 由QA组织所执行的一种检查，用以保证项目采取了良好的质量管理准则。

质量保证组织 以在不同要点处评审项目和产品确保实施良好的质量管理准则为主要目的并且永久设立的组织或小组。同时也在所支持的项目上为测试提供帮助及相关提交内容。此QA组织必须依靠项目团队。

质量改进 改变方法以降低缺陷率。

质量管理 过程及方法的执行，用以保证作为开发过程输出的质量。

质量计划 规划质量方法。

质量控制 将产品质量与标准相比较的过程。

中期修复 在下一个正式版本发布前的软件修复工作，但不是立即（例如，一周左右）。

走查 用来分析技术工作成果的测试技术。

参 考 文 献

1. Andrews, M. and Whittaker, J. A. 2006. *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Upper Saddle River, NJ: Pearson Education.
2. Arthur, L. J. 1993. *Improving Software Quality: An Insider's Guide to TQM*. New York: Wiley.
3. Beck, K. 2002. *Test Driven Development: By Example*. New York: Addison-Wesley.
4. Beizeir, B. 1991. *Software Testing Techniques* (Second Edition). New York: Van Nostrand Reinhold.
5. Beizeir, B. 1995. *Black-Box Testing: Techniques for Functional Testing for Testing of Software and Systems*. New York: Wiley.
6. Brooks, F. P., Jr. 1995. *The Mythical Man-Month*, Anniversary Edition. Reading, MA: Addison- Wesley.
7. Card, D. N. with Robert, L. G. 1990. *Measuring Software Design Quality*. Englewood Cliffs, NJ: Prentice Hall.
8. Charette, R. N. 1990. *Applications Strategies for Risk Analysis*. New York: McGraw-Hill.
9. Cho, C. K. 1987. *Quality Programming: Developing and Testing Software with Statistical Quality Control*. New York: Wiley.
10. Copeland, L. 2003. *A Practitioner's Guide to Software Test Design*. Norwood, MA: Artech House.
11. Creech, B. 1994. *The Five Pillars of TQM*. New York: Truman Valley/Dutton.
12. Davis, A. M. 1990. *Software Requirements: Analysis and Specification*. Englewood Cliffs, NJ: Prentice Hall.
13. Davis, B. 1994. *The Economics of Automatic Testing* (Second Edition), New York: McGraw-Hill.
14. DeCarlo, N. J. and Kent, S. W. 1990. History of the Malcolm Baldrige National Quality Award. *Quality Progress* (March): 21-27.
15. Deming, W. E. 1986. *Out of the Crisis*. Cambridge, MA: Massachusetts Institute of Technology, Center for Advanced Engineering Study.
16. Deming, W. E. *The Deming Management Method*. New York: Perigee.
17. Fagan, M. E. 1986. Advances in software inspections. *IEEE Transactions on Software Engineering* (July): 744-751.
18. Fournier, G. 2008. *Essential Software Testing: A Use-Case Approach*. New York: Taylor & Francis.
19. Freedman, D. P. and Weinberg, G. M. 1990. *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Third Edition. New York: Dorset House.
20. Giles, A. C. 1992. *Software Quality: Theory and Management*. New York: Chapman & Hall.
21. Glass, R. L. 1991. *Software Conflict-Essays on Art and Science of Software Engineering*. Englewood Cliffs, NJ: Yourdon (Prentice Hall).
22. Grady, R. B. 1989. Dissecting software failures. *HPJournal* (April): 57-63.
23. Grady, R. B. 1992. *Practical Software Metrics for Project Management and Process Improvement*. Englewood

- Cliffs, NJ: Prentice Hall.
24. Gub, T. et al. 1993. *Software Inspection*. Reading, MA: Addison-Wesley.
 25. Hahn, G. J. 1995. Deming's impact on industrial statistics: some reflections. *The American Statistician*, 49, 4 (November): 336-341.
 26. Hatton, L. 1997. Re-examining the defect density versus component size distribution. *IEEE Software* (March/April).
 27. Hetzel, B. 1988. *The Complete Guide to Software Testing* (Second Edition). Wellesley, MA: QED.
 28. Hetzel, B. 1993. *Making Software Measurement Work: Building an Effective Measurement Program*. Wellesley, MA: QED.
 29. Hollocker, C. P. 1990. *Software Reviews and Audits Hand Book*. New York: Wiley.
 30. Humphrey, W. S. 1989. *Managing the Software Process*. Reading, MA: Addison-Wesley.
 31. Humphrey, W. S. 1995. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley.
 32. *IEEE Standard for Software Verification and Validation Plans*. 1986. Washington, DC: IEEE, Std. 1012-1986 (R1992).
 33. *IEEE Standard for Measures to Produce Reliable Software*. 1988. Washington, DC: IEEE, Std. 982.
 34. *IEEE Standard Glossary of Software Engineering Terminology*. 1990. Washington, DC: IEEE, Std. 610.12-1990.
 35. Jarvis, A. S. 1988. How to establish a successful test plan. *EDP Quality Assurance Conference*, Washington, DC, November 14-17.
 36. Jarvis, A. S. 1994. Applying Software Quality. *The Seventh International Software Quality Week*, San Francisco, May 17-20.
 37. Jarvis, A. S. 1995a. Applying Metrics. *First World Congress for Software Quality Conference*, San Francisco, June 20-22.
 38. Jarvis, A. S. 1995b. Exploring the Needs of a Developer and Tester, *Quality Conference 95*, Santa Clara, CA, April 4-7.
 39. Jones, C. 1991. *Applied Software Management: Assuring Productivity and Quality*. New York: McGraw-Hill.
 40. Jones, C. 1993. *Assessment and Control of Software Risks*. Englewood Cliffs, NJ: Yourdon Press Computing Services.
 41. Jones, C. 1993. *Critical Problems in Software Measurement*. Carlsbad, CA: Infosystems Management.
 42. Kaner, C., Falk, J., and Nguyen, H. Q. 1999. *Testing Computer Software*. Hoboken, NJ: John Wiley & Sons.
 43. Lewis, R. O. 1992. *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software*. New York: Wiley.
 44. Lewis, W. E. 1998. Spiral Testing. *Quality Assurance Institute Annual International Information Technology Quality Conference*, April 13-17, Orlando, FL.
 45. Lewis, W. E. 2003. *PDCA/Test*. Auerbach: Boca Raton, FL.
 46. Lewis, W. E. 2004. *Software Testing and Continuous Quality Improvement* (Second Edition). Auerbach: Boca Raton, FL.
 47. Marciniak, J. 1994. *Encyclopedia of Software Engineering*. New York: Wiley.
 48. Marks, D. M. 1992. *Testing Very Big Systems*. New York: McGraw-Hill.
 49. Martin, J. 1989. *Information Engineering Book I Introduction*. Englewood Cliffs, NJ: Prentice Hall.

50. Martin, J. 1990a. *Information Engineering Book II Planning and Analysis*. Englewood Cliffs, NJ: Prentice Hall.
51. Martin, J. 1990b. *Information Engineering Book III Design and Construction*. Englewood Cliffs, NJ: Prentice Hall.
52. Martin, J., Kathleen, K. C., and Joe, L. 1991. *Systems Application Architecture: Common User Access*. Englewood Cliffs, NJ: Prentice Hall.
53. McCabe, J. J. and Butler, C. W. 1989. Design complexity measurement and testing. *Communications of the ACM*, 32, 12 (December): 1415-1424.
54. McCabe, T. J. 1982. *Structured Testing: A Software Testing Methodology Using Cyclomatic Complexity Metric*. National Bureau of Standards Special Publication, December: 500-599.
55. McConnell, S. 1993. *Code Complete: A Practical Handbook of Software Construction*. Redmond, WA: Microsoft.
56. Murine, G. E. 1988. Integrating software quality metrics with software QA. *Quality Progress* (November): 38-43.
57. Musa, J. D., Iannino, A., and Okumoto, K. 1987. *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill.
58. Myers, G. G., Sandier, C., Badgett, T., and Thomas, T. 2004. *The Art of Software Testing* (Second Edition). New York: Wiley.
59. Orr, K. 1981. *Structured Requirements Definition*. Topeka, KS: Orr.
60. Page-Jones, M. 1985. *Practical Project Management: Restoring Quality to DP Projects and Systems*. New York: Dorset House.
61. Patton, R. 2005. *Software Testing* (Second Edition). Indianapolis, IN: Sam's.
62. Perry, W. E. 1986. *How to Test Software Packages: A Step-by-Step Guide to Assuring They Do What You Want*. New York: Wiley.
63. Perry, W. E. 1991. *Quality Assurance for Information Systems: Methods, Tools, and Techniques*. Wellesley, MA: QED.
64. Perry, W. 2006. *Effective Methods for Software Testing*. Hoboken, NJ: John Wiley & Sons.
65. Pettichord, B., Kaner, 2001. C., Bach, J., *Lessons Learned in Software Testing: A Context Driven Approach*. Hoboken, NJ: John Wiley & Sons.
66. Pressman, R. S. 1992. *Software Engineering: A Practitioner's Approach, Third Edition*. New York: McGraw-Hill.
67. Roper, M. 1993. *Software Testing*. New York: McGraw-Hill.
68. Royer, T. C. 1992. *Software Testing Management: Life on the Critical Path*. Englewood Cliffs, NJ: Prentice Hall.
69. Rubin, H. 1993. *Practical Guide to the Design and Implementation of IS Measurement Programs*. Englewood Cliffs, NJ: Prentice Hall.
70. Sanders, J. 1994. *Software Quality: A Framework for Success in Software Development*. Reading, MA: Addison-Wesley.
71. Schulmeyer, G. G. 1990. *Zero Defect Software*. New York: McGraw-Hill.
72. Schulmeyer, W. G. and McManus, J. 1992. *Total Quality Management for Software*. New York: Van Nostrand Reinhold.

73. Sharp, A. 1993. *Software Quality and Productivity*. New York: Van Nostrand Reinhold.
74. Spillner, A., Schaefer, Hans, Linz, Tilo. 2006. *Software Testing Foundations*. Sebastopol, CA: Rocky Nook.
75. Vinay, P. 2008. *Managing Software Testing*. New York: CRC Press.
76. Tripathy, P. and Naik, S. 2008. *Software Testing and Quality Assurance: Theory and Practice*. Hoboken, NJ: John Wiley & Sons.
77. Ward, P. T. and Stephen, J. M. 1985a. *Structured Development for Real Time Systems. Vol. 1: Introduction and Tools*. Englewood Cliffs, NJ: Yourdon.
78. Walton, M. 1986. *The Deming Management Method*. New York: A Perigee Book.
79. Weinberg, G. M. 1992. *Quality Software Management: Systems Thinking, Vol. 1*. New York: Dorset House.
80. Weinberg, G. M. 1992. *Software Quality Management: Vol. 1: Systems Thinking*. New York: Dorset House.
81. Weinberg, G. M. 1993. *Quality Software Management: First-Order Measurement, Vol. 2*. New York: Dorset House.
82. Weinberg, G. M. 1993. *Software Quality Management: Vol. 2: First-Order Measurement*. New York: Dorset House.
83. Whittaker, J. 2002. *How to Break Software: A Practical Guide to Testing*. Upper Saddle River, NJ: Pearson.
84. Whittaker, J. and Thompson, H. 2003. *How to Break Software Security: Effective Techniques for Security Testing*. Upper Saddle River, NJ: Pearson.
85. Yourdon, E. 1985. *Structured Walkthroughs*, Third Edition. Englewood Cliffs, NJ: Yourdon.

索引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

- Acceptance testing (验收测试), 257
 - approvals, obtaining (验收测试获得批准), 258
 - complete acceptance test cases (完成验收测试用例), 256-257
 - complete acceptance test planning (完成验收测试计划), 253-256
 - conducting (确定), 253-259
 - defect types (缺陷类型), 268-269
 - documenting acceptance defects (记录验收缺陷), 259
 - environment, establishing (建立验收测试环境), 256
 - executing (执行验收测试), 258-259
 - new acceptance tests, executing (执行新验收测试), 259
 - plan, reviewing/approving (计划、评审/批准), 257-258
 - regression test, acceptance fixes (验收修复回归测试), 258-259
 - review, scheduling (评审、进度安排), 257-258
 - schedule, finalizing (完成验收测试进度安排), 255
 - system-level test cases, identification of (系统级测试用例的识别), 257
 - team, organizing (组建验收测试团队), 255-256
 - tools, installing (安装验收测试工具), 256
 - types, finalizing (完成验收测试类型), 253-255
- Ad hoc testing (即兴测试), 68-71, 318-319
 - advantages, disadvantages of (即兴测试的优缺点), 71
 - art of (即兴测试的艺术), 68-71
- Agenda, defining (定义议程), 156
- Agile testing (敏捷测试), 371-376
 - acceptance testing (验收测试), 451
 - agile planning (敏捷计划), 372-374
 - compliance testing (合规性测试), 375-376
 - continuous improvement (持续改进), 444
 - formal requirements, agile user stories, contrasted (正式需求与敏捷用户故事对比), 371-372
 - information gathering (信息收集), 445
 - methodology (方法论), 137-276, 443-452
 - preparing for next spiral (准备下一次螺旋测试), 449
 - Quick Test Professional, 374
 - spiral test results, summarizing/reporting (总结/报告螺旋测试结果), 452
 - system testing (系统测试), 450
 - test case design (测试用例设计), 447
 - test development (测试开发), 448
 - Test-Driven Development (测试驱动开发), 374
 - test execution/evaluation (测试执行/评价), 448
 - test planning (测试计划), 446
 - types of (敏捷测试的类型), 374-375
 - user story, defining (定义用户故事), 372
- Ambiguity review checklist (歧义性评审检查表), 536-537
- Application GUI components, identification of (确定应用图形用户界面组件), 202
- Application-specific function libraries (特定于应用的函数库), 336
- Approval procedures, defining (定义审批规程), 187-188
- Approvals, obtaining (获得批准), 194, 206-208
- Architecture review checklist (架构评审检查表), 538
- Attributes testing checklist (属性测试检查表), 514-516
- Automated testing (自动化测试), 399-400, 410-439, 492
 - acceptance test (验收测试), 437
 - acquisition plan, developing (开发自动化测试获取计划), 432, 434
 - assessment (评估), 323-342
 - candidate review, conducting (进行候选工具评审), 433
 - candidate tools, identification of (确定候选工具), 433
 - candidates, scoring (为候选工具打分), 433
 - determine whether goals have been met (确定是否满足目标), 439
 - evaluation methodology (评估方法), 431-439
 - evaluation plan, creation of (创建评估计划), 436
 - framework (框架), 334-342, 382
 - implement modifications (实施修改), 438
 - maturity (成熟度), 329-330
 - operating environment, tools in (自动化测试中工具的运

行环境), 438-439
 orientation, conducting (召开推介会议), 437-438
 procure testing tool (采购测试工具), 436
 proposals, solicitation of (简化建议), 435
 receive tool (接收工具), 437
 request for proposal, generation of (生成请求建议), 434-435
 requirements, reviewing (评审需求), 434
 selection activities for formal procurement, conducting (正式采购模式下的选择活动), 434-435
 selection activities for informal procurement, conducting (非正式采购模式下的选择活动), 432-434
 selection criteria, defining (定义选择标准), 432-433
 selection of tool (工具的选择), 434
 set tool objectives (设定工具目标), 432
 standard frameworks (标准框架), 337-339
 technical evaluation, performing (进行技术评估), 435
 technical requirements document, creating (创建技术需求文档), 434
 test requirements, defining (定义测试需求), 431
 tool identification (确定测试自动化工具), 332-333
 tool in operating environment (运行环境中的工具), 438-439
 tool manager's plan, creation of (制定工具经理的计划), 436-437
 tool selection (工具选择), 434
 tool source, selection of (选择工具来源), 435
 training plan, creation of (创建培训计划), 437
 training tool users (培训工具的使用者), 438
 write evaluation report (撰写评估报告), 439
 Automation maturity (自动化成熟度), 329-330

B

Baldrige, Malcolm, 34
 Baldrige performance excellence criteria (波多里奇的绩效优秀标准), 35
 Basis path testing (基本路径测试), 557-558
 Black-box testing (黑盒测试), 39-40, 558-559
 extra program logic (附加的程序逻辑), 559
 Bottom-up testing (自底向上测试), 559
 Boundary value testing (边界值测试), 559-561
 field ranges (字段范围), 560
 GUI (图形用户界面), 561
 nonnumeric input data (非数值输入数据), 560
 nonnumeric output data (非数值输出数据), 561
 number of outputs (输出数量), 561
 numeric input data (数值输入数据), 560
 numeric output data (数值输出数据), 560
 output range of values (值输出的范围), 560

 tables or arrays (表或数组), 560-561
 Branch/condition coverage testing (分支/条件覆盖测试), 562-563
 Branch coverage testing (分支覆盖测试), 561-562
 BTO, 参见Business technology optimization
 Building test plan (建立测试计划), 168-188
 Burnout tracking (零缺陷跟踪), 228-231, 264-266
 Business technology optimization (企业技术优化), 6

C

Capability maturity model (能力成熟度模型), 29-33
 Capability Maturity Model for Software (软件能力成熟度模型), 34
 Cause-effect graphing (画因果图), 563-567
 causes (原因), 565
 decision table (决策表), 566
 effects (结果), 565-567
 methodology (方法), 564
 specification (规格说明), 564-567
 Change request form (变更请求格式), 457
 Change request procedures, establishing (建立变更请求规程), 184-185
 Checklists (检查表), 493-556
 requirements phase defect checklist (需求阶段缺陷检查表), 493-494
 China, emergence of software companies in (软件公司在中国出现), 387
 Clarification request, template (澄清请求模板), 484-485
 Client/server challenge (客户/服务器架构的挑战), 140-141
 Client/server spiral testing, psychology of (客户/服务器螺旋测试的心理学), 141-146
 integration of QA, development (把质量保证和开发结合起来), 143-144
 iterative/spiral development methodology (迭代/螺旋式开发方法), 144-146
 new school of thought (新思想), 141-142
 tester/developer perceptions (对测试人员/开发人员的理解), 142-143
 CMM, 参见Capability maturity model
 CMMI, 33-34
 Coding phase defect checklist (编码阶段缺陷检查表), 499-502
 Commercial vendor tool descriptions (商用厂商工具描述), 410
 Compliance testing (合规性测试), 364-365, 375-376
 Computer risk analysis (计算机风险分析), 163
 Computer Society of Institute of Electrical and Electronics Engineers, guidelines for writing software requirements specifications (IEEE计算机学协会的编写软件需求规约

- 指导原则), 371-372
- Condition coverage (条件覆盖), 567-568
- Configuration build procedures, defining (定义配置构建规程), 186
- Continuous competency development (持续能力提升), 382
- Continuous improvement phased approach (“阶段化”持续改进方法), 88
- Continuous improvement spiral testing (持续改进螺旋测试), 151-154
- Continuous quality improvement (持续质量改进), 280-281
- Control flow testing checklist (控制流程测试检查表), 523
- Control testing checklist (控制测试检查表), 518-523
- CRUD testing (CRUD测试), 568
- Cultural differences (文化差异), 394
- ## D
- Data design review checklist (数据设计评审检查表), 539-540
- Data-driven framework (数据驱动框架), 338
- Data generation strategies (数据生成策略), 401-408
- cutting-edge test case generator, requirements-based (基于需求的有效测试用例生成器), 404-408
 - data based on database, generating (根据数据库生成数据), 403-404
 - sampling from production (生产数据抽样), 401-402
 - seeding data (数据播种), 402-403
 - starting from scratch (从零开始), 402
- Database testing (数据库测试), 568-600
- attributes (属性)
 - definition (定义), 584
 - vs. relationships (属性与关系), 586
 - column key integrity (列键完整性), 569-570
 - columns (列), 574
 - compound primary keys (复合主键), 575
 - CRUD testing (CRUD测试), 569
 - customer address table (客户地址表), 589
 - customer/salesperson table (客户/销售人员表), 584
 - customer table (客户表), 588-589, 595
 - data modeling essentials (数据建模基础), 571-573
 - database integrity testing (数据库完整性测试), 568-571
 - definition (定义), 594
 - dependency constraints constraint rule (依赖约束约束规则), 597-600
 - desk table (办公桌表), 578
 - domain (域), 584-585
 - domain integrity (域完整性), 570
 - domain names (域名), 585-588
 - employee/project table (员工/项目表), 581-582
 - employee table (员工表), 578-579, 581-582, 587, 598-599
 - entities, definition (实体的定义), 574-575
 - entities vs. relationships (实体与关系), 583-584
 - entity classes (实体类), 577
 - entity integrity (实体完整性), 568
 - entity subtypes (实体子类型), 594
 - first normal form (第一范式), 588-589
 - identification (识别), 575-577
 - identifying entities (定义实体), 576-577
 - line-item table (行项表), 597
 - many-to-many (多对多), 580
 - model, defining (定义模型), 572
 - model creation, rationale (模型创建基本原理), 572-573
 - model refinement (模型提炼), 593
 - model use, in database design (在数据库设计中使用模型), 599
 - multiple relationships (多重关系), 582-583
 - normalization (规范化), 588-597
 - defined (定义), 586-587
 - null values (空值), 575-576
 - one-to-many (一对多), 579-580
 - one-to-one (一对一), 578-579
 - order (次序), 574
 - order table (订单表), 592-593, 595, 597
 - primary key integrity (主键完整性), 569
 - problems with unnormalized entities (非规范化实体的问题), 587-588
 - product/model/contract table (产品/型号/合同表), 593
 - product model table (产品型号表), 590-591
 - product table (产品表), 591, 595
 - project table (项目表), 581-582
 - purchase agreement table (购买协议表), 583
 - recursion (递归), 598-599
 - referential integrity (参照完整性), 570-571, 594-597
 - referential integrity test cases (参照完整性测试用例), 571
 - relational design (关系型设计), 600
 - relationship types (关系类型), 577-578
 - relationships, definition (关系定义), 577-584
 - rows (行), 574
 - sample table (表样本), 574
 - second normal form (第二范式), 590-591
 - table names (表名称), 573-574
 - tables, definition (表的定义), 573-574
 - telephone line table (电话线路表), 580
 - third normal form (第三范式), 591-593
 - user-defined integrity (用户定义的完整性), 570
- Decision tables (决策表), 600-601
- decision table (决策表), 601
- Defect gap analysis (缺陷差距分析), 228, 264

- Defect management process (缺陷管理过程), 301-307
- defect category (缺陷种类), 303
 - defect discovery, classification (缺陷发现与分类), 301-302
 - defect meetings (面对缺陷), 305
 - defect metrics (缺陷度量), 305-306
 - defect priority (缺陷优先级), 302-303
 - defect summary (缺陷总结), 304-305
 - defect tracking (缺陷跟踪), 303-304
 - defect reporting (缺陷报告), 304
 - quality control (质量控制), 301
 - quality standards (质量标准), 306-307
- Defect recording/tracking procedures, establishing (建立缺陷记录/跟踪规程), 182-183
- Defect report, template (缺陷记录模板), 470-472, 490
- Defect severity status (缺陷严重性状态), 228, 264
- Defects, method of finding (找出缺陷的方法), 266-267
- Defining metric objectives, test planning (定义度量目标, 测试计划), 188-193
- Defining system/acceptance tests (定义系统/验收测试), 203-206
- Deming, Edward
- contribution of (Edward Deming的贡献), 75-76
 - fourteen quality principles of (14条质量原则), 77-83
 - adoption of new philosophy (采用新的质量观念), 78
 - awarding business on price tag alone (仅靠价签来激励企业), 79
 - barriers between staff areas, breaking down (打破员工区域之间的壁垒), 81
 - constancy of purpose (始终如一的目标), 77-78
 - education (教育), 82
 - fear, driving out (驱除恐惧), 80-81
 - institute leadership (确立领导职责), 80
 - mass inspection, ceasing dependence on (停止对大量审查的依赖), 78-79
 - numerical codes, elimination of (驱除数字化目标), 81-82
 - pride of workmanship, removing barriers to (消除阻碍员工自豪感的壁垒), 82
 - production, service, improving (改进生产和服务), 79
 - retraining (再培训), 79-80, 82
 - slogans, exhortations, targets for workforce, elimination of (解除工作口号、训词及目标), 81
 - training (培训), 79-80
 - transformation, taking action to accomplish (采取行动完成转变), 82-83
- Dependencies, defining (定义依赖关系), 177-178
- Design, reviewing/approving (测试设计的评审和批准), 206-208
- Design and execution maturity (测试设计和执行成熟度), 328-329
- Desk checking (桌面检查), 601
- Development acceptance, obtaining (获得开发人员认可), 25-26
- Development methodology overview (开发方法概述), 139-154
- client/server challenge (客户/服务器架构的挑战), 140-141
 - continuous improvement spiral testing (持续改进螺旋测试方法), 151-154
 - joint application designs, role of (JAD的角色), 146
 - limitations of life-cycle development (生命周期开发的局限性), 139-140
 - prototypes, methodology for developing (开发原型的方法), 148-151
 - demonstrating prototype to users (向用户演示原型), 150
 - production system, developing (开发产品系统), 151
 - prototype, developing (开发原型), 148-149
 - prototypes to management, demonstrating (向管理层演示原型), 149
 - specifications, revising and finalizing (修订并定稿规格), 150-151
 - prototyping, role of (原型的角色), 146-148
 - psychology of client/server spiral testing (客户/服务器螺旋测试的心理学), 141-146
 - integration of QA, development (把质量保证和开发的结合起来), 143-144
 - iterative/spiral development methodology (迭代/螺旋式开发方法), 144-146
 - new school of thought (新思想), 141-142
 - tester/developer perceptions (对测试人员/开发人员的理解), 142-143
- Division of responsibilities (职责划分), 316-317
- Dollar estimation (经济评估), 163-164
- Dynamic testing code, static testing (代码的静态测试和动态测试), 131-136
- ## E
- Effort estimation (工作量估算)
- maturity (成熟度), 328
 - model project (项目建模), 294-296
- Elements of software configuration management (软件配置管理的要素), 20
- Emerging specialized areas in testing (软件测试中的新型专业领域), 321-396
- Environment readiness checklist (环境准备检查表), 529-530

Equivalence partitioning (等价划分), 601-604

equivalence class partitioning, test cases using (使用等价类划分创建测试用例), 603-604

field ranges (字段范围), 602

income/tax test cases (收入/税金测试用例), 602

income vs. tax percentage (收入与税金百分比), 602

nonnumeric input data (非数值输入数据), 603

nonnumeric output data (非数值输出数据), 603

number of items (项数), 603

number of outputs (输出数), 603

numeric input data (数值输入数据), 602

numeric output data (数值输出数据), 602

output range of values (输出值的范围), 602

sets of values (值集), 602

tables or arrays (表或数组), 603

Error handling, defining (定义错误处理), 337

Error testing checklist (错误测试检查表), 506-508

Establish transparency (建立透明性), 394

Estimating test work effort (测试工作量估算), 292-293

Evaluation of automated testing tools (自动化测试工具评估), 431-439

acquisition plan, developing (开发自动化测试获取计划), 432, 434

candidate review, conducting (进行候选工具评审), 433

candidate tools, identification of (确定候选工具), 433

candidates, scoring (为候选工具打分), 433

determine whether goals have been met (确定是否满足目标), 439

evaluation plan, creation of (创建评估计划), 436

implement modifications (实施修改), 438

orientation, conducting (召开推介会议), 437-438

perform acceptance test (执行验收测试), 437

procure testing tool (采购测试工具), 436

proposals, solicitation of (简化建议), 435

receive tool (接收工具), 437

request for proposal, generation of (生成请求建议), 434-435

requirements, reviewing (评审需求), 434

selection activities for formal procurement, conducting (正式采购模式下的选择活动), 434-435

selection activities for informal procurement, conducting (非正式采购模式下的选择活动), 432-434

selection criteria, defining (定义选择标准), 432-433

set tool objectives (设定工具目标), 432

technical evaluation, performing (进行技术评估), 435

technical requirements document, creating (创建技术需求文档), 434

test requirements, defining (定义测试需求), 431

tool in operating environment (运行环境中的工具),

438-439

tool manager's plan, creation of (制定工具经理的计划), 436-437

tool selection (工具选择), 434

tool source, selection of (选择工具来源), 435

training plan, creation of (创建培训计划), 437

training tool users (培训工具的使用者), 438

write evaluation report (撰写评估报告), 439

Evolution of automated testing tools (自动化测试工具的发展), 8-11

Exception testing (异常测试), 604-605

test case/error exception test matrix (测试用例/错误异常测试矩阵), 605

Exploratory testing (探索性测试), 72-73

advantages, disadvantages of (探索性测试的优缺点), 73

art of (探索性测试的艺术), 72

Extreme programming (极限编程), 8

F

Factors limiting testing tools (限制测试工具使用的因素), 429-430

Field testing checklist (字段测试检查表), 502-503

File test checklist (文件测试检查表), 505-506

Final test report, publishing (发布最终测试报告), 273-276

Final test summary report, template (最终测试总结报告模板), 491-492

First computers, development of (第一台计算机的开发), 7

Folder structure, defining (定义文件夹结构), 335-336

Formal requirements, agile user stories, contrasted (敏捷用户故事与正式需求对比), 371-372

FORTRAN, first IGL programming language (第一代编程语言FORTRAN), 7

Foundation for Malcolm Baldrige National Quality Award (美国波多里奇国家质量奖的基础), 34-37

Free-form testing (自由形式测试), 605

Function, defects by (按功能分类缺陷), 264

Function/test matrix (功能/测试矩阵)

building (构建), 200

template (模板), 464

Function tests, designing (设计功能测试), 195-200

Functional specification review checklist (功能规约评审检查表), 540-546

Functional test requirements, refining (完善功能测试规约), 195-199

Functions tested and not tested (功能已测试/未测试), 267-268

G

Goals of usability testing (易用性测试的目的), 359-364

accessibility testing (可访问性测试), 361-364
 guidelines for usability testing (易用性测试的原则), 361
 Section 508 (第508条), 361-364
 Gray-box testing (灰盒测试), 41, 605-606
 GUI-based functional test matrix, template (基于图形用户界面的功能测试矩阵模板), 465
 GUI component test matrix, template (图形用户界面组件测试矩阵模板), 464
 GUI design, guidelines (图形用户界面的指导原则), 200-201
 GUI tests (图形用户界面测试)
 defining (定义), 202-203
 designing (设计), 200-203

H

High-level business requirements, identification of (确定总体业务需求), 161-162
 High-level functional requirements, defining (定义总体功能需求), 170
 High-level project activities, identification of (确定高级项目活动), 292
 Histograms (直方图), 606-607
 response time histogram (响应时间直方图), 606
 response time of 100 samples (100个样本的响应时间), 606
 History of software testing (软件测试的历史), 3-11
 business technology optimization (业务技术优化), 6
 development of first computers (第一代计算机的开发), 7
 evolution of automated testing tools (自动化测试工具的发展), 8-11
 extreme programming (极限编程), 8
 FORTRAN, first IGL programming language (第一代编程语言FORTRAN), 7
 historical software testing and development parallels (历史上软件测试和开发并行), 6-8
 popular scripting techniques (流行的脚本技术), 11
 static capture/replay tools with scripting language (具备脚本语言的静态捕获/回放工具), 10
 static capture/replay tools without scripting language (不带脚本语言的静态捕获/回放工具), 10
 testing principles (测试原则), 5
 variable capture/replay tools (可变的捕获/回放工具), 10-11

I

Identifying high-level project activities (确定高级项目活动), 292
 Impact analysis checklist (影响分析检查表), 527-528

India, emergence of software companies in (软件公司在印度出现), 387
 Individual finding, defects by (按发现者分类缺陷), 267
 Industry best processes (行业最佳过程), 381
 Information gathering (信息收集), 155-165
 agenda, defining (定义议程), 156
 high-level business requirements, identification of (确定总体业务需求), 161-162
 interview (访谈)
 conducting (执行), 156-165
 confirming findings (确认访谈成果), 165
 preparing for (准备), 156
 summarizing (总结), 165
 participants, identification of (确定参加访谈的人), 156
 project, understanding (理解项目), 158-159
 project development methodology, understanding (理解项目开发方法), 161
 project objectives, understanding (理解项目目标), 159-160
 project plans, understanding (理解项目计划), 160-161
 project status, understanding (理解项目状态), 160
 risk analysis, performing (进行风险分析), 162-165
 summarize findings (总结成果), 165
 Inspections (审查), 102-103, 607
 Instinct (本能), 163
 Integrated Product Development Capability Maturity Model (集成产品开发能力成熟度模型), 34
 Integrated testing, development (测试与开发的整合), 309-313
 development methodology, modifying (修改开发方法), 312
 incorporate defect recording (缺陷记录合并), 313
 integrated team (整合后的团队), 313
 quality control (质量控制), 309
 select integration points (选择整合时间点), 311-312
 tasks to integrate, identification of (确定要整合的任务), 310-311
 test methodology training (测试方法培训), 312-313
 test steps, tasks, customizing (定制测试步骤及任务), 311
 test team, organizing (组件测试团队), 310
 Integration testing (集成测试)
 completeness of integration test conditions, evaluation of (评估集成测试条件的完整性), 124-125
 creating cases (创建集成测试用例), 122-123
 integration test conditions, creation of (创建集成测试条件), 124
 interfaces for completeness, reconciliation of (全面协调接口), 124
 methodology for (集成测试方法), 123

unit interfaces, identifying (标识出单元接口), 123-124
 Interim report, publishing (发布中期报告), 220-221
 International Standards Organization (国际标准化组织), 29
 Interviews (访谈)
 conducting (执行), 156-165
 confirming findings (确认访谈成果), 165
 preparing for (准备), 156
 summarizing (总结), 165
 IPD-CMM, 参见Integrated Product Development Capability Maturity Model
 ISO, 参见International Standards Organization
 ISO9000, 29
 Iterative/spiral development methodology (迭代/螺旋式开发方法), 144-146

J

JADs, 参见Joint application designs
 Joint application designs (联合应用设计), 40, 608
 role of (JAD的角色), 146

K

Knowledge acquisition process (知识获取过程), 345-346

L

Life-cycle testing, psychology of (生命周期测试的心理学), 89
 Limitations of life-cycle development (生命周期开发的局限性), 139-140
 Load testing (负载测试), 344
 Logical design phase defect checklist (逻辑审计阶段缺陷检查表), 494-495

M

Maintenance process, defining (定义维护过程), 337
 Malcolm Baldrige National Quality Award (美国波多里奇国家质量奖), 34-37
 Management acceptance, obtaining (获得管理层认可), 25
 Management overhead (管理开销), 394
 Manual/automated GUI/function tests, scripting (开发手工/自动化图形用户界面/功能测试脚本), 209-210
 Manual/automated new spiral tests, executing (执行新的螺旋测试的手工/自动化测试), 219
 Manual/automated system fragment tests, scripting (开发手工/自动化阶段性系统测试脚本), 210
 Manual/automated test types, identification of (确定手工/自动化测试的类型), 171
 Manual vs. automated testing (手工测试与自动化测试), 41

Match/merge checklist (匹配/合并检查表), 511-512
 Methodology checklist (方法检查表), 109-110
 Methodology development (开发方法), 139-154
 client/server challenge (客户/服务器架构的挑战), 140-141
 continuous improvement spiral testing (持续改进螺旋测试), 151-154
 joint application designs, role of (JAD的角色), 146
 limitations of life-cycle development (生命周期开发的局限性), 139-140
 prototypes, methodology for developing (开发原型的方法), 148-151
 demonstrating prototype to users (向用户演示原型), 150
 production system, developing (开发产品系统), 151
 prototype, developing (开发原型), 148-149
 prototypes to management, demonstrating (向管理层演示原型), 149
 specifications, revising and finalizing (修订并定稿规约), 150-151
 psychology of client/server spiral testing (客户/服务器螺旋测试的心理学), 141-146
 integration of QA, development (把质量保证和开发结合起来), 143-144
 iterative/spiral development methodology (迭代/螺旋式开发方法), 144-146
 new school of thought (新思想), 141-142
 tester/developer perceptions (对测试人员/开发人员的理解), 142-143
 role of prototyping (原型法的作用), 146-148
 Metric points, defining (定义度量要点), 189-193
 Minutes of meeting, template (会议纪要), 476-478
 Modern software testing tools (现代软件测试工具), 397-440
 Modular framework (模块化框架), 338-339

N

National Institute of Standards and Technology (美国国家标准与技术研究院), 34-37
 New school of thought (新思想), 141-142
 NIST, 参见National Institute of Standards and Technology
 Nonexistent, poor requirements (需求不存在或编写粗劣), 68-73
 Nonexistent requirements (需求不存在), 68-73
 Nonfunctional testing (非功能测试), 343-365
 compliance testing (合规性测试), 364-365
 goals of usability testing (易用性测试的目的), 359-364
 accessibility testing (易用性测试), 361-364
 guidelines for usability testing (易用性测试的原则),

361

Section 508 (第508条), 361-364

knowledge acquisition process (知识获取过程), 345-346

load testing (负载测试), 344

performance deliverables (性能测试可交付物), 350-351

performance monitoring (性能监视), 344

performance testing (性能测试), 343-345

security testing (安全性测试), 351-353

file integrity checkers (文件完整性检查器), 356-357

log reviews (日志评审), 356

network scanning (网络扫描), 353-354

password cracking (密码破译), 355-356

penetration testing (渗透测试), 357-358

scope of security testing, identifying (确定安全性测试范围), 352

test case generation, execution (生成测试用例并执行), 353

types of (类型), 353-358

virus detectors (病毒检测器), 357

vulnerability scanning (漏洞扫描), 354-355

stress testing (压力测试), 344

test development (测试开发), 346-350

usability testing (易用性测试), 358-359

volume testing (容量测试), 344

Numerical method for evaluating requirement quality (评估需求质量的数值方法), 54-55

O

OAT, 参见Orthogonal array testing

On-site/offshore model (本土/离岸模型), 383-395

challenges (挑战), 393-394

economic trade-offs, determining (确定经济上的得失), 384

future of (本土/离岸模型的未来), 394-395

implementing (实现), 388-389

application management (应用管理), 389

detailed design (详细设计), 388

knowledge transfer (知识转移), 388

milestone-based transfer (基于里程碑的转移), 388-389

steady state (稳定状态), 389

methodology, benefits of (本土/离岸方法的收益), 392-394

outsourcing methodology (外包方法), 385-388

offshore activities (离岸活动), 387-388

on-site activities (本土活动), 386-387

prerequisites (先决条件), 389-392

relationship model (关系模型), 389-391

standards (标准), 391-392

selection criteria, determining (确定选择标准), 385

Open-source freeware vendor tools (开源自由件厂商工具), 410

Operating model (运营模型), 381-382

Organizational architecture (组织架构), 315

Organizational relationships (组织关系), 317

Orthogonal array testing (正交表测试), 608-610

parameter combinations (参数组合), 610

parameter combinations with total enumeration (带完整列举的参数组合), 609

Outsourcing methodology (外包方法), 385-388

offshore activities (离岸活动), 387-388

on-site activities (本土活动), 386-387

Overview of testing techniques (测试技术概述), 39-50

black-box testing (黑盒测试), 39-40

gray-box testing (灰盒测试), 41

joint application designs (JAD), 40

manual vs. automated testing (手工测试与自动化测试), 41

static vs. dynamic testing (静态测试与动态测试), 41-42

taxonomy of software testing techniques (软件测试方法的分类), 42-50

white-box testing (白盒测试), 40

P

Parallels in development, software testing (软件测试与开发并行), 6-8

Pareto analysis (帕累托分析法), 610-611

Participant roles (参与人员角色), 103-105

Participants, identification of (确定参加的人), 156

PDCA, 参见Plan, do, check, act

People Capability Maturity Model (人员能力成熟度模型), 33

Performance deliverables (性能测试可交付物), 350-351

Performance monitoring (性能监视), 344

Performance testing (性能测试), 343-345

Physical design phase defect checklist (物理设计阶段缺陷检查表), 496-498

Placeware, 388

Plan, do, check and act (计划、执行、检查、改进), 84, 137-276

continuous improvement through (通过计划、执行、检查、改进实现持续改进), 83-84

test schedule, template (测试进度表模板), 481

Plan for review process (规划评审过程), 105

Poor requirements (需求编写粗劣), 68-73

Popular scripting techniques (流行的脚本技术), 11

Positive and negative testing (正反测试), 611-612

Potential acceptance tests, identification of (确定可能的验收测试), 206

- Potential system tests, identification of (确定可能的系统测试), 203-205
- Preparation for next spiral (准备下一次螺旋测试), 223-231
- acceptance tests, updating (更新验收测试), 225
 - function/GUI tests, updating (更新功能/图形用户界面), 223-225
 - metric graphics, publishing (发布度量图), 227-231
 - procedures, reassessing (重新评价规程), 225-227
 - publishing interim test report (发布中期测试报告), 227-231
 - refine tests (细化测试), 223-225
 - system fragment tests, updating (更新阶段性系统测试), 225
 - team (团队)
 - procedures, and test environment, reassessing (重新评价规程和测试环境), 225-227
 - reassessing (重新评价), 225-227
 - test control procedures, reviewing (评审测试控制规程), 226-227
 - test environment (测试环境)
 - reassessing (重新评价), 225-227
 - updating (更新), 227
 - test team, evaluating (评价测试团队), 225-226
- Prevention vs. detection (预防与检测), 14-15
- Prior defect history testing (缺陷历史预测试), 612
- Procedures testing checklist (规程测试检查表), 517
- Process evaluation methodology (过程评估方法), 324-330
- analyzing information (分析信息), 325-326
 - analyzing test maturity (分析测试成熟度), 326-330
 - documenting findings (记录结果), 330
 - gathering information (收集信息), 325-326
 - identify key elements (标识关键元素), 324-325
 - presenting findings (呈现结果), 330
- Process for creating test cases from good requirements (根据好的需求创建测试用例的过程), 55-64
- name test cases (命名测试用例), 59-62
 - requirements, reviewing (评审需求), 55-58
 - test case descriptions and objectives, writing (编写测试用例描述及目标), 62
 - test cases (测试用例)
 - creating (创建), 62-63
 - reviewing (评审), 63-64
 - test plan, writing (编写测试计划), 58
 - test suite, identifying (确定测试套件), 58-59
- Product quality and project quality (产品质量和项目质量), 279-280
- Product scope and project scope (产品范围和项目范围), 283-284
- Program unit design phase defect checklist (程序单元设计阶段缺陷检查表), 498-499
- Project, understanding (理解项目), 158-159
- Project charter (项目规章), 284
- Project completion checklist (项目完成情况检查表), 530-532
- Project development methodology, understanding (理解项目开发方法), 161
- Project framework (项目框架), 281-283, 318-319
- components of (项目框架的组成), 280
 - continuous quality improvement (持续质量改进), 280-281
 - executing, monitoring, and controlling phases (执行、监视和控制阶段), 282-283
 - implement phase (实现阶段), 283
 - initiation phase (启动阶段), 281-282
 - planning phase (计划阶段), 282
 - where no quality infrastructure exists (在质量基础设施不存在的情况下), 317-318
- Project goal, integration of QA, development (项目的目标: 把质量保证和开发结合起来), 143-144
- Project information gathering checklist (项目信息收集检查表), 525-527
- Project issue resolution procedures, defining (定义项目问题解决规程), 186-187
- Project management framework (项目管理框架), 279-290
- benefits of (项目管理框架的好处), 290
 - product quality and project quality (产品质量和项目质量), 279-280
 - product scope and project scope (产品范围和项目范围), 283-284
 - project charter (项目规章), 284
 - project framework, components of (项目框架的组成), 280
 - project framework and continuous quality improvement (项目框架与持续质量改进), 280-281
 - project framework phases (项目框架的各个阶段), 281-283
 - executing, monitoring, and controlling phases (执行、监视和控制阶段), 282-283
 - implement phase (实现阶段), 283
 - initiation phase (启动阶段), 281-282
 - planning phase (计划阶段), 282
 - project manager in quality management, role of (项目经理在质量管理中的作用), 285-286
 - scope statement (范围陈述), 285
 - scoping project to ensure product quality (确定项目范围以确保产品质量), 283
 - summarizing/reporting tests results (总结/报告测试结果), 279

- test manager advice (给测试经理的建议), 288-289
 - business knowledge, updating (更新业务知识), 289
 - communicate issues as they arise (出现问题时及时沟通), 288-289
 - improve process (改进过程), 289
 - knowledge base creation (创建知识库), 289
 - new testing technologies and tools, learning (学习新的测试技术和测试工具), 289
 - requesting help from others (请求别人的帮助), 288
 - test manager in quality management role (测试经理在质量管理中的作用), 286-288
 - analyzing requirements (分析需求), 286
 - analyzing test results (分析测试结果), 288
 - duplication and repetition, avoiding (避免重复), 287
 - perform gap analysis (进行差距分析), 286-287
 - quality (质量), 288
 - test data, defining (定义测试数据), 287
 - validation of test environment (确认测试环境), 287-288
 - Project management methodology (项目管理方法), 277-320
 - Project manager in quality management, role of (项目经理在质量管理中的作用), 285-286
 - Project objectives, understanding (理解项目目标), 159-160
 - Project plans, understanding (理解项目计划), 160-161
 - Project quality management (项目质量管理), 291-299
 - effort estimation, model project (工作量估算: 项目建模), 294-296
 - estimating test work effort (测试工作量估算), 292-293
 - identifying high-level project activities (确定高级项目活动), 292
 - project quality management processes (项目质量管理过程), 291
 - quality planning (质量计划), 292
 - quality standards (质量标准), 296-299
 - test planning (测试计划), 293-294
 - Project status, understanding (理解项目状态), 160
 - Project status report, template (项目状态报告模板), 486-488
 - Prototypes (原型)
 - demonstrating prototype to users (向用户演示原型), 150
 - demonstrating to management (向管理层演示原型), 149
 - developing (开发原型), 148-149
 - methodology for developing (开发原型的方法), 148-151
 - production system, developing (开发产品系统), 151
 - review checklist (评审检查表), 546-547
 - specifications, revising, finalizing (修订并定稿规约), 150-151
 - Prototyping (原型法), 612-617
 - application prototyping (应用程序原型法), 615
 - cyclic models (循环模型), 613
 - data-driven prototyping (数据驱动原型法), 616
 - early-stage prototyping (早期原型法), 617
 - evolutionary and throwaway (演化和舍弃), 615
 - fourth-generation languages and prototyping (第四代语言和原型法), 614
 - iterative development accounting (迭代开发对账), 614
 - prototype systems development (原型系统开发), 615-616
 - replacement of traditional life cycle (传统生命周期的替代), 616
 - role of (角色), 146-148
 - user software engineering (用户软件工程), 617
 - Psychology of client/server spiral testing (客户/服务器螺旋测试的心理学), 141-146
 - integration of QA, development (把质量保证和开发结合起来), 143-144
 - iterative/spiral development methodology (迭代/螺旋式开发方法), 144-146
 - new school of thought (新思想), 141-142
 - tester/developer perceptions (对测试人员/开发人员的理解), 142-143
 - Psychology of life-cycle testing (生命周期测试的心理学), 89
 - Public Company Accounting Reform and Investor Protection Act of 2002, 参见 Sarbanes-Oxley Act
 - Public Law 100-107 (100-107公法), 34-37
 - Publishing final test report (发布最终测试报告), 273-276
 - Publishing interim report (发布中期报告), 220-221, 227-231
 - Publishing metric graphics (发布度量图), 227-231
- ## Q
- QTR, 参见 Quick Test Professional
 - Quality, defined (定义质量), 13-14
 - Quality assurance components (质量保证的组成), 17-18
 - software testing (软件测试), 17-18
 - Quality assurance framework (质量保证框架), 13-37
 - Baldrige, Malcolm, 34
 - Baldrige performance excellence criteria (波多里奇绩效优秀标准), 35
 - capability maturity model (能力成熟度模型), 29-33
 - Capability Maturity Model for Software (软件能力成熟度模型), 34
 - CMMI, 33-34
 - Foundation for Malcolm Baldrige National Quality Award (美国波多里奇国家质量奖的基础), 34-37
 - Integrated Product Development Capability Maturity Model (集成产品开发能力成熟度模型), 34
 - International Standards Organization (国际标准化组织), 29

- ISO9000, 29
- Malcolm Baldrige National Quality Award (美国波多里奇国家质量奖), 34-37
- National Institute of Standards and Technology (美国国家标准与技术研究院), 34-37
- People Capability Maturity Model (人员能力成熟度模型), 33
- prevention vs. detection (预防与检测), 14-15
- Public Law 100-107 (100-107公法), 34-37
- quality, defined (定义质量), 13-14
- quality assurance components (质量保证的组成), 17-18
 - software testing (软件测试), 17-18
- quality control (质量控制), 18-23
 - software configuration management (软件配置管理), 19-23
- quality standards (质量标准), 26-37
- Sarbanes-Oxley Act (《萨班斯-奥克斯利法案》), 26-29
- software quality assurance (软件质量保证), 16-17
- software quality assurance plan (软件质量保证计划), 23-25
- software quality assurance plan, developing/implementing (开发和实施软件质量保证计划), 23-26
- Systems Engineering Capability Model (系统工程能力成熟度模型), 34
- total quality management program (总的质量管理程序), 29
- verification vs. validation (验证与确认), 15-16
- Quality control (质量控制), 18-23
 - software configuration management (软件配置管理), 19-23
- Quality planning (质量计划), 292
- Quality principles of Deming (Deming的质量原则), 77-83
- Quality standards (质量标准), 26-37, 296-299
- Quality through continuous improvement process (质量的持续改进过程), 75-84
 - continuous improvement through plan, do, check, act process (通过计划、执行、检查、改进实现持续改进), 83-84
- Deming, Edward
 - contribution of (Edward Deming的贡献), 75-76
 - fourteen quality principles of (14条质量原则), 77-83
 - plan, do, check, act circles (计划、执行、检查、改进), 84
- statistical methods, role of (统计方法扮演的角色), 76-77
 - case-and-effect diagram (因果图), 76
 - control chart (控制图), 77
 - flowchart (流程图), 76
 - histogram (直方图), 77
 - Pareto chart (帕累托图), 76-77
 - run chart (运行图), 77
 - scatter diagram (散布图), 77
- Quick Test Professional, 374
- R**
- Random testing (随机测试), 618
- Range testing (范围测试), 618
- Record testing checklist (记录测试检查表), 503-505
- Recovery functions, defining (定义恢复函数), 337
- Regression testing (回归测试), 618-620
 - establishing strategy (建立回归测试策略), 172-174
 - manual/automated spiral fixes (回归测试手工/自动化修复螺旋过程中发现的缺陷), 217-219
 - maturity (成熟度), 329
 - range testing test cases (范围测试的测试用例), 619
- Reporting procedures, establishing (建立报告规程), 187
- Reporting progress (进度报告), 319-320
- Reporting tests results (报告测试结果), 261-276
 - approvals, obtaining (获得批准), 273
 - final test report (最终测试报告)
 - preparing for (准备), 263-272
 - publishing (发布), 273-276
 - reviewing/approving (评审和批准), 272-276
 - findings/recommendations, developing (总结测试结果/建议), 269-272
 - metric graphics, analyzing/creating (分析/创建度量图), 263-269
 - perform data reduction (执行数据精简), 261-262
 - project framework (项目框架), 279
 - project overview, preparing (准备项目概述), 263
 - remaining defects to matrix, posting (将剩余的缺陷写入一个矩阵), 262
 - review, scheduling/conducting (评审的日程安排/执行), 272-273
 - test activities, summarizing (总结测试活动), 263
 - test defects by test number, consolidating (按测试编号整理测试缺陷), 261-262
 - tests executed/resolved, ensuring (确保测试均已执行/解决), 261
- Requirement changes, identification of (确定需求变更), 221
- Requirement quality factors (需求质量因素), 52-54
 - modifiable (可修改), 53
 - necessary (必需), 53
 - nonredundant (非冗余), 53
 - within scope (在范围内), 54
 - terse (简洁), 54
 - testable (可测试), 54
 - traceable (可跟踪), 54
 - understandable (可理解), 52-53

- Requirements, transforming to testable test cases (将需求转换成可测试的测试用例), 51-73
- ad hoc testing (即兴测试), 68-71
- exploratory testing (探索性测试), 72-73
- nonexistent, poor requirements (需求不存在或编写粗劣), 68-73
- numerical method for evaluating requirement quality (评估需求质量的数值方法), 54-55
- process for creating test cases from good requirements (根据好的需求创建测试用例的过程), 55-64
- name test cases (命名测试用例), 59-62
- requirements, reviewing (评审需求), 55-58
- test case descriptions and objectives, writing (编写测试用例描述及目标), 62
- test plan, writing (编写测试计划), 58
- test suite, identifying (确定测试套件), 58-59
- requirement quality factors (需求质量因素), 52-54
- modifiable (可修改), 53
- necessary (必需), 53
- nonredundant (非冗余), 53
- within scope (在范围内), 54
- terse (简洁), 54
- testable (可测试), 54
- traceable (可跟踪), 54
- understandable (可理解), 52-53
- software requirements as basis of testing (将软件需求作为测试的基础), 51-52
- transforming use cases to test cases (将用例转换为测试用例), 64-68
- generating test cases (生成测试用例), 66-68
- summary (小结), 68
- test data, generating (生成测试数据), 68
- use case diagram, drawing (绘制用例图), 64
- use case scenarios, identifying (确定用例场景), 66
- use case text, writing (编写用例文本), 64-66
- Requirements definition maturity (需求定义成熟度), 326-327
- Requirements review checklist (需求评审检查表), 547-551
- Requirements specification (需求规约), 455-456
- Requirements traceability matrix, template (需求可追溯矩阵模板), 461-462
- Retest matrix, template (重测试矩阵模板), 474-475
- Reusing generic functions, application-specific function libraries (复用通用函数和特定于应用的函数库), 336
- Review, scheduling (评审的日程安排)
- conducting (执行), 194
- preparing for (准备), 206, 210-212
- Review agenda, developing (开发评审纪要), 106
- Review report, creating (创建评审报告), 106
- Reviewing test planning (评审测试计划), 194
- Reviews types (评审类型), 101-103
- Risk analysis, performing (进行风险分析), 162-165
- Risk-based testing (基于风险的测试), 620
- Robustness, modularize scripts/test data to increase (模块化脚本/测试数据以增加健壮性), 11,336
- Root cause analysis (根本原因分析), 266
- Run charts (运行图), 620-621
- ## S
- Sahi, 374
- Sample run chart (运行图示例), 621
- Sandwich testing, 621
- Sarbanes-Oxley Act (《萨班斯-奥克斯利法案》), 26-29
- Schedule review, 105
- Scope statement (范围陈述), 285
- Scoping project to ensure product quality (为确保产品质量划定项目范围), 283
- Screen data mapping, template (屏幕数据映射模板), 485
- Scripting guidelines and review checklists, developing (开发脚本的指导原则和评审检查表), 336-337
- Search test checklist (搜索测试检查表), 510-511
- SECM, 参见 Systems Engineering Capability Model
- Security design strategy (安全性设计策略), 242-243
- Security testing (安全性测试), 351-353
- file integrity checkers (文件完整性检查器), 356-357
- log reviews (日志评审), 356
- network scanning (网络扫描), 353-354
- password cracking (密码破译), 355-356
- penetration testing (渗透测试), 357-358
- scope of security testing, identifying (安全性测试), 352
- test case generation, execution (生成测试用例并执行), 353
- types of (类型), 353-358
- virus detectors (病毒检测器), 357
- vulnerability scanning (漏洞扫描), 354-355
- SEI-CMM, 参见 Software Engineering Institute-Capability Maturity Model
- Selenium, 374
- Service Oriented Architecture testing (面向服务架构的测试), 367-369
- SOA testing (SOA测试), 参见 Service Oriented Architecture testing
- Software Engineering Institute-Capability Maturity Model (软件工程研究所能力成熟度模型), 29-33
- Software licensing (软件许可证), 394
- Software quality (软件质量), 1-84
- Software quality assurance plan (软件质量保证计划), 16-17, 23-25, 453-454

- executing (执行), 26
- implementation planning (实施计划), 26
- software quality assurance plan, developing/implementing (开发和实施软件质量保证计划), 23-26
- Software requirements as basis of testing (将软件需求作为测试的基础), 51-52
- Software test plan (软件测试计划), 92-93
- Software testing as continuous improvement process (将软件测试作为持续改进过程), 89-92
- Software testing techniques (软件测试技术), 557-628
- Software testing trends (软件测试趋势), 399-408
 - automated capture/replay testing tools (自动捕获/回放测试工具), 399-400
 - data generation strategies (数据生成策略), 401-408
 - cutting-edge test case generator, requirements-based (基于需求的有效测试用例生成器), 404-408
 - data based on database, generating (根据数据库生成数据), 403-404
 - sampling from production (生产数据抽样), 401-402
 - seeding data (数据播种), 402-403
 - starting from scratch (从零开始), 402
 - necessary and sufficient conditions (必要条件和充分条件), 400-401
 - test case builder tools (测试用例构建工具), 400
- Southeast Asia, emergence of software companies in (软件公司出现在东南亚), 387
- Spiral software testing methodology (螺旋软件测试方法), 137-276
- Spiral test defects, documenting (用文档记录螺旋测试缺陷), 219
- Spiral testing methodology (螺旋测试方法), 443-452
 - acceptance testing (验收测试), 451
 - continuous improvement (持续改进), 444
 - information gathering (信息收集), 445
 - preparing for next spiral (准备下一次螺旋测试), 449
 - spiral test results, summarizing/reporting (总结/报告螺旋测试结果), 452
 - system testing (系统测试), 450
 - test case design (测试用例设计), 447
 - test development (测试开发), 448
 - test execution/evaluation (测试执行/评价), 448
 - test planning (测试计划), 446
- Spiral testing summary report, template (螺旋测试总结报告模板), 476
- SQA, 参见Software quality assurance plan
- State transition testing (状态转换测试), 622-623
- Statement coverage testing (语句覆盖测试), 622
- States testing checklist (状态测试检查表), 516-517
- Static capture/replay tools (静态捕获/回放工具)
 - with scripting language (具备脚本语言), 10
 - without scripting language (不附带脚本语言), 10
- Static testing (静态测试)
 - dynamic testing, contrasted (与动态测试相对), 41-42
 - dynamic testing code (动态测试代码), 131-136
 - acceptance testing (验收测试), 134-135
 - defect recording (缺陷记录), 135-136
 - executing test plan (执行测试计划), 132-133
 - integration testing (集成测试), 134
 - system testing (系统测试), 134
 - testing coding with technical reviews (带技术评审的测试编码), 131-132
 - unit testing (单元测试), 133
 - logical design (逻辑设计), 115-119
 - data model, process model, and linkage (数据模型、过程模型和链接), 115-117
 - refining system/acceptance test plan (细化系统/验收测试计划), 118-119
 - testing logical design with technical reviews (带技术评审的测试逻辑设计), 117-118
 - physical design (物理设计), 121-125
 - completeness of integration test conditions (集成测试条件的完整性), 124-125
 - integration test cases, creating, (创建集成测试用例) 122-123
 - interfaces for completeness, reconciliation of (全面协调接口), 124
 - methodology for (方法), 123
 - technical reviews, testing physical design with (通过技术评审测试物理设计), 121-122
 - test conditions, creation of (创建测试条件), 124
 - unit interfaces, identifying (标识出单元接口), 123-124
 - program unit design (程序单元设计), 127-129
 - creating unit test cases (创建单元测试用例), 128-129
 - with technical reviews (带技术评审的), 127-128
- requirements (需求), 107-113
 - with ambiguity reviews (歧义性评审), 108-109
 - inspections (审查), 109-110
 - requirements traceability matrix (需求可追溯矩阵), 110-111
 - system/acceptance test plan, building (建立系统/验收测试计划), 111-113
 - with technical reviews (技术评审), 109
 - walkthroughs (走查), 109-110
- Statistical methods, role of (统计方法扮演的角色), 76-77
 - cause-and-effect diagram (因果图), 76
 - control chart (控制图), 77
 - flowchart (流程图), 76
 - histogram (直方图), 77

- Pareto chart (帕累托图), 76-77
 - run chart (运行图), 77
 - scatter diagram (散布图), 77
 - Statistical profile testing (统计概况测试), 623
 - Strategy maturity (策略成熟度), 327-328
 - Stress testing (压力测试), 344
 - checklist (检查表), 513-514
 - Structured walkthroughs (结构化走查), 101-102, 623-625
 - state transition table (状态转换表), 624
 - Summarizing/reporting tests results (总结/报告测试结果), 261-276
 - approvals, obtaining (获得批准), 273
 - final test report (最终测试报告)
 - preparing for (准备), 263-272
 - publishing (发布), 273-276
 - reviewing/approving (评审/审批), 272-276
 - findings/recommendations, developing (总结测试结果/建议), 269-272
 - metric graphics, analyzing/creating (分析/创建度量图), 263-269
 - perform data reduction (执行数据精简), 261-262
 - project overview, preparing (准备项目概述), 263
 - remaining defects to matrix, posting (将剩余的缺陷写入一个矩阵), 262
 - review, scheduling/conducting (评审的日程安排/执行), 272-273
 - test activities, summarizing (总结测试活动), 263
 - test defects by test number, consolidating (按测试编号整理测试缺陷), 261-262
 - tests executed/resolved, ensuring (确保测试均已执行/解决), 261
 - Syntax testing (语法测试), 625
 - System/acceptance test plan, template (系统/验收测试计划模板), 460-461
 - System fragment tests, designing (设计阶段性系统测试), 205-206
 - System summary report, template (系统总结报告模板), 469-470
 - System testing (系统测试), 233-252
 - approving (审批), 250-251
 - backup tests, designing/scripting (设计/脚本化备份测试), 247
 - compatibility tests, designing/scripting (设计/脚本化兼容性测试), 244-245
 - complete system test cases (完成系统测试案例), 239-250
 - complete system test plan (完成系统测试计划), 233-239
 - conversion tests, designing/scripting (设计/脚本化转换测试), 245-246
 - defect types (缺陷类型), 268
 - environment, establishing (搭建系统测试环境), 238-239
 - executing (执行系统测试), 251-252
 - installation tests, designing/scripting (设计/脚本化安装测试), 248-249
 - monitoring (监视), 240-241
 - new system tests, executing (执行新的系统测试), 251
 - other system test types, designing/scripting (设计/脚本化其他类型的系统测试), 249-250
 - performance tests, design/script (设计/脚本化性能测试), 239-240
 - probe (探测法), 241
 - recovery tests, designing/scripting (设计/脚本化可恢复性测试), 248
 - regression test, system fixes (对系统测试中的修正进行回归测试), 251
 - reviewing (评审), 250-251
 - schedule, finalizing (确定系统测试日程安排), 235
 - script documentation tests, design/ (设计/脚本化文档测试), 246-247
 - security tests, designing/scripting (设计/脚本化安全性测试), 242-243
 - stress tests, designing/scripting (设计/脚本化压力测试), 243-244
 - system defects, documenting (将系统测试缺陷记录在文档中), 251-252
 - team, organizing (组建团队), 235-238
 - test drivers (测试驱动模块), 241
 - tools, installing (安装工具), 239
 - types, finalizing (确定系统测试类型), 233-235
 - usability tests, designing/scripting (设计/脚本化易用性测试), 246
 - volume tests, designing/scripting (设计/脚本化容量测试), 243
 - Systems Engineering Capability Model (系统工程能力成熟度模型), 34
- T**
- Table testing (表测试), 625-626
 - Taxonomy, software testing tools (软件测试工具的分类), 409-430
 - commercial vendor tool descriptions (商业厂商工具描述), 410
 - factors limiting testing tools (限制测试工具的因素), 429-430
 - open-source freeware vendor tools (开源自由件厂商工具), 410
 - test automation (测试自动化), 410-430
 - testing tool selection checklist (测试工具选择检查表), 409-410

- Taxonomy of software testing techniques (软件测试技术的分类), 2-50
- TDD, 参见Test-Driven Development
- Technical design review checklist (技术设计评审检查表), 552-554
- Technical reviews (技术评审)
 - as continuous improvement process (将技术评审作为持续改进过程), 96-100
 - motivation (动机), 101
- Test approvals, template (测试审批模板), 478
- Test automation (测试自动化), 410-430, 492
- Test automation assessment (测试自动化评估), 323-342
 - identifying applications to automate (标识需要自动化的应用), 332
 - identifying best test automation tool (确定最佳测试自动化工具), 332-333
 - test execution (测试执行), 333-334
 - test script maintenance (测试脚本维护), 334
 - test scripting (编写测试脚本), 333
- Test automation framework (测试自动化框架), 334-342, 382
 - automation framework, basic features (自动化框架的基本特性), 335-337
 - hybrid framework (混和框架), 341-342
 - keyword-driven framework (关键字驱动框架), 11, 339-341, 635
 - standard automation frameworks (标准自动化框架), 337-339
- Test automation maturity (测试自动化成熟度), 329-330
- Test automation standard frameworks (测试自动化标准框架), 337-339
- Test automation strategy, template (测试自动化策略模板), 492
- Test automation tool identification (确定测试自动化工具), 332-333
- Test case builder tools (测试用例构建工具), 400
- Test case design (测试用例设计), 195-208
 - application GUI components, identification of (确定应用图形用户界面组件), 202
 - approvals, obtaining (获得批准), 206-208
 - defining system/acceptance tests (定义系统/验收测试), 203-206
 - design, reviewing/approving (设计的评审和批准), 206-208
 - function/test matrix, building (建立功能/测试矩阵), 200
 - function tests, designing (设计功能测试), 195-200
 - functional test requirements, refining (完善功能测试需求), 195-199
 - GUI tests (通行用户界面设计), 200-201
 - defining (定义), 202-203
 - designing (设计), 200-203
 - potential acceptance tests, identification of (确定可能的验收测试), 206
 - potential system tests, identification of (确定可能的系统测试), 203-205
 - review, scheduling/preparing for (评审的日程安排/准备), 206
 - system fragment tests, designing (设计阶段性系统测试), 205-206
- Test case execution status (测试用例执行状态), 227-228
- Test case log, template (测试用例日志模板), 466-467
- Test case preparation review checklist (测试用例准备评审检查表), 554-555
- Test case template (测试用例模板), 466
- Test checklist (测试检查表), 508-509
- Test condition vs. test case, template (测试条件与测试用例模板), 486
- Test coverage through traceability (通过可追溯性实现测试覆盖), 213-216
- Test defect details report, template (测试缺陷详细报告模板), 489
- Test deliverables, defining (定义测试可交付物), 174-175
- Test development (测试开发), 209-212, 346-350
 - approving (批准), 210-212
 - manual/automated GUI/function tests, scripting (开发手工/自动化图形用户界面/功能测试脚本), 209-210
 - manual/automated system fragment tests, scripting (开发手工/自动化阶段性系统测试脚本), 210
 - reviewing (评审), 210-212
 - test scripts, developing (开发测试脚本), 209-210
- Test-Driven Development (测试驱动开发), 374
- Test environment, establishing (搭建测试环境), 177
- Test execution/evaluation (测试执行/评价), 217-221
 - evaluation (评价), 219-220
 - manual/automated new spiral tests, executing (执行新的螺旋测试的手工/自动化测试), 219
 - metrics, analyzing (分析度量标准), 219-220
 - publishing interim report (发布中期报告), 220-221
 - regression test, manual/automated spiral fixes (回归测试手工/自动化修复上次螺旋过程中的缺陷), 217-219
 - requirement changes, identification of (确定需求变更), 221
 - setup (组织测试内容), 217-219
 - spiral test defects, documenting (记录螺旋测试中发现的缺陷), 219
 - test schedule, refining (细化测试进度表), 220-221
 - testing (测试), 217-219
- Test execution plan, template (测试执行计划模板), 479

- Test execution tracking manager, template (测试执行跟踪经理模板), 490
- Test exit criteria, identification of (确定测试退出标准), 171-172
- Test log summary report, template (测试日志总结报告模板), 468
- Test management constraints (测试管理的约束), 315-320
- ad hoc testing (即兴测试), 318-319
 - division of responsibilities (职责划分), 316-317
 - organizational architecture (组织架构), 315
 - organizational relationships (组织关系), 317
 - project framework (项目框架), 318-319
 - project framework where no quality infrastructure exists (在质量基础设施不存在的情况下的项目框架), 317-318
 - reporting progress (进度报告), 319-320
 - traceability/validation matrix (可追溯性/确认矩阵), 319
 - traits of well-established quality organization (已建立得很好的质量保证部门的特征), 315-316
- Test manager (测试经理), 286-289
- analyzing requirements (分析需求), 286
 - analyzing test results (分析测试结果), 288
 - business knowledge, updating (更新业务知识), 289
 - communicate issues as they arise (出现问题时的沟通), 288-289
 - duplication and repetition, avoiding (避免重复), 287
 - improve process (改进过程), 289
 - knowledge base creation (创建知识库), 289
 - new testing technologies and tools, learning (学习新的测试技术和测试工具), 289
 - perform gap analysis (完成差距分析), 286-287
 - quality (质量), 288
 - requesting help from others (请求别人帮助), 288
 - test data, defining (定义测试数据), 287
 - validation of test environment (确认测试环境), 287-288
- Test plan components (测试计划的组成), 95-96
- Test plan template (测试计划模板), 462-463
- Test planning (测试计划), 167-194, 293-294
- approval procedures, defining (定义审批规程), 187-188
 - approving (审批), 95, 194
 - building test plan (构建测试计划), 168-188
 - change request procedures, establishing (建立变更请求的规程), 184-185
 - configuration build procedures, defining (定义配置构建规程), 186
 - defect recording/tracking procedures, establishing (建立缺陷记录/跟踪规程), 182-183
 - defining metric objectives (定义度量目标), 188-193
 - dependencies, defining (定义依赖关系), 177-178
 - development (开发), 93-95
 - high-level functional requirements, defining (定义总体功能需求), 170
 - manual/automated test types, identification of (确定手工/自动化测试的类型), 171
 - metric points, defining (定义度量要点), 189-193
 - metrics, defining (定义度量), 188-189
 - preparing (准备), 168-170
 - project issue resolution procedures, defining (定义项目问题解决规程), 186-187
 - regression test strategy, establishing (建立回归测试策略), 172-174
 - reporting procedures, establishing (建立报告规程), 187
 - review, scheduling/conducting (评审的日程安排/执行), 194
 - reviewing (评审), 95, 194
 - schedule test (进度表测试), 95
 - specifications development (规约开发), 95
 - test deliverables, defining (定义测试可交付物), 174-175
 - test environment (测试环境)
 - defining (定义), 95
 - establishing (建立), 177
 - test exit criteria, identification of, 171-172
 - test objectives, defining (定义测试目标), 93
 - test schedule, creating (创建测试计划), 178
 - test team, organizing (组件测试团队), 175-176
 - test tools, selection of (选择测试工具), 178-182
 - version control procedures, establishing (建立版本控制规程), 185-186
- Test process assessment (测试过程评估), 323-324
- Test project milestones, template (测试项目里程碑模板), 480
- Test schedule (项目进度表)
 - creating (创建), 178
 - refining (细化), 220-221
 - template (模板), 472-473
- Test scripts, developing (开发测试脚本), 209-210
- Test strategy, template (测试策略模板), 481-484
- Test team, organizing (组建测试团队), 175-176
- Test templates (测试模板), 459-492
- clarification request (澄清请求), 484-485
 - defect report (缺陷报告), 470-472, 490
 - final test summary report (最终测试总结报告), 491-492
 - function/test matrix (功能/测试矩阵), 464
 - GUI-based functional test matrix (基于图形用户界面的功能测试矩阵), 465
 - GUI component test matrix (图形用户界面组件测试矩阵), 464
 - minutes of meeting (会议纪要), 476-478

- plan, do, check, act test schedule (PDCA测试进度表), 481
- project status report (项目状态报告), 486-488
- requirements traceability matrix (需求可追溯性矩阵), 461-462
- retest matrix (再测试矩阵), 474-475
- screen data mapping (屏幕数据映射), 485
- spiral testing summary report (螺旋测试总结报告), 476
- System/acceptance test plan (系统/验收测试计划), 460-461
- system summary report (系统总结报告), 469-470
- test approvals (测试审批), 478
- test automation strategy (测试自动化策略), 492
- test case (测试用例), 466
- test case log (测试用例日志), 466-467
- test condition vs. test case (测试条件和测试用例), 486
- test defect details report (测试缺陷详细报告), 489
- test execution plan (测试执行计划), 479
- test execution tracking manager (测试执行追踪管理表), 490
- test log summary report (测试日志总结报告), 468
- test plan (测试计划), 462-463
- test project milestones (测试项目里程碑), 480
- test schedule (测试进度表), 472-473
- test strategy (测试策略), 481-484
- unit test plan (单元测试计划), 459-460
- Test tools, selection of (测试工具的选择), 178-182
- Tester, defects by (按测试人员分类缺陷), 264
- Tester/developer perceptions (对测试人员/开发人员的理解), 142-143
- Testing center of excellence (测试卓越中心), 377-382
- continuous competency development (持续能力提升), 382
- industry best processes (行业最佳过程), 381
- operating model (运营模型), 381-382
- test automation framework (测试自动化框架), 382
- testing metrics (测试矩阵), 381
- Testing metrics (测试矩阵), 381
- Testing physical design with technical reviews (通过技术评审测试物理设计), 121-122
- Testing principles (测试原则), 5
- Testing tool selection checklist (测试工具选择检查表), 409-410, 524-525
- Thread testing (线程测试), 626
- Top-down testing (自顶向下测试), 626-627
- Total quality management program (全面质量管理计划), 29
- TQM program (TQM计划), 参见Total quality management program
- Traceability, test coverage through (通过可追溯性实现测试覆盖), 213-216
- Traceability/validation matrix (可追溯性/确认矩阵), 319
- Traits of well-established quality organization (已建立得很好的质量保证部门的特征), 315-316
- Transforming requirements to testable test cases (将需求转换为可测试的测试用例), 51-73
- Transforming use cases to test cases (将用例转换为测试用例), 64-68
- ## U
- Unit test plan, template (单元测试计划模板), 459-460
- Unit testing checklist (单元测试检查表), 532-535
- Usability testing (易用性测试), 358-359
- Use cases, transforming to test cases (将用例转换为测试用例), 64-68
- generating test cases (生成测试用例), 66-68
- summary (小结), 68
- test data, generating (生成测试数据), 68
- use case diagram, drawing (绘制用例图), 64
- use case scenarios, identifying (确定用例场景), 66
- use case text, writing (编写用例文本), 64-66
- ## V
- Variable capture/replay tools (可变的捕获/回放工具), 10-11
- Verification vs. validation (验证与确认), 15-16
- Version control procedures, establishing (建立版本控制规程), 185-186
- Volume testing (容量测试), 344
- ## W
- Waterfall development methodology (瀑布式开发方法), 87-88
- Waterfall testing review (瀑布测试评审), 85-136
- Watir, 374
- Web applications, functional testing tools (Web应用的功能测试工具), 374
- Weighting risk attributes (风险属性加权), 164-165
- White-box testing (白盒测试), 40, 627
- regression test acceptance fixes (回归测试验收修复), 258-259
- regression test manual/automated spiral fixes (回归测试手工/自动化修复上次螺旋过程中的缺陷), 217-219
- ## X
- XP, 参见Extreme programming